CHALMERS

User-friendly reconfiguration of AGV layouts

Master's Thesis in Systems, Control and Mechatronics

Martin Dahl, Anna Forsberg

Department of Signals and Systems
Automatic control, Automation and Mechatronics
Chalmers University of Technology
In collaboration with Agve

Gothenburg, Sweden 2014
Master's Thesis 2014:1

Abstract

Automated Guided Vehicles (AGVs) are an important part of the industry today, performing a wide range of tasks, including transport, lifting and cleaning. Most modern AGVs travel on predefined paths in a layout, using wireless navigation such as lasers or inertial systems.

The reasons for why an AGV's predefined path might need changing are many. For instance a leak from the roof might occur just above the path of the AGV, a pallet crate might be (mis)placed between two nodes or another AGV might break down while dealing with some task. Today only one way of dealing with this kind of problems exists: an engineer needs to be invited to reconfigure the path. This might take time and money and is one of the main reasons why smaller companies refrain from utilizing AGVs.

This project presents an application that allows any operator on site to redefine the path between two consecutive nodes. The application requires the operator to drive the vehicle around the obstacle in an appropriate way. The operator might sample data along the way, which is not necessary, but gives more options later on. The application analyses the driven path and sample data (if such exists) and suggests 1-4 different paths.

The paths are created using combinations of clothoids, NURBS, B-splines and straight lines, giving every path different properties. The paths are calculated within a few seconds, are driveable, merge with the path prior as well as after the new segment and allow the AGV to drive at high speed when appropriate.

As the application is unaware of the character of the obstacle it is up to the operator to choose the path best suited in the specific situation. By choosing one of the paths the operator changes the predefined route of the AGV between these nodes and the newly generated path will be used until further notice.

Acknowledgements

This project was performed in cooperation with AGVE. We would like to thank them not only for coming up with an interesting topic, but also providing us with an office to work in, an AGV to try out our ideas on and a great supervisor, knowledgeable and very helpful (Jonas, your help was very much appreciated!). On top of that all the personnel at the office were always friendly and supportive, which earns them our sincere thank you!

We would also like to acknowledge the help our academical adviser gave us. Always ready to contribute with new ideas, listen to our thoughts and, not to disregard, read our report. Thank you very much, Petter!

And of course this would not have gone as smoothly and the results would not have been as great if it wasn't for all the support we got from our respective partner. Even though neither of you had the patience to read all of our report, your encouragement along the way meant a lot. Thank you!

Anna & Martin, Göteborg 20/9-14

Contents

List of Algorithms

List of Figures

1 Introduction

Automated Guided Vehicles (AGVs) have been used in different forms since 1955. Early AGVs used a wire buried in the floor as the guidance system, while modern vehicles use wireless navigation utilizing lasers (Laser Guided Vehicles, LGVs) or inertial systems[1]. AGVs that are no longer bound to a wire are called free-ranging. AGVE has customers using all kinds of solutions, but for new installations LGVs are most common.

Present-day's computers installed on AGVs allow the guide paths to be stored on-board. This means new stations can be added in a more flexible way and the configuration of the guide paths is no longer a hardware problem, instead software is used to define paths[2]. While a central traffic management system oversees the different AGVs to avoid collisions and queues, the individual vehicle acts as an autonomous agent.

AGVs have for several decades been, and still are, an area of active research. Many possible uses for AGVs in industries have already been discovered and explored, and more applications are coming up every year. Therefore the number of AGVs in industries is constantly increasing, especially in larger companies such as Volvo and Samsung.

A company such as Volvo usually does not need to remodel the paths of AGVs very often and is content to have well planned predefined paths stored in the software of every vehicle. If the need for redesign of a path would emerge, a large company generally has engineers on site to take care of it.

However, the need of an engineer might arise when no engineer is available, for instance if in the middle of the night an AGV breaks down, blocking the path for other vehicles. The entire production-line comes to a standstill if AGVs can't deliver and money is lost. Situations like this are not uncommon and are one of the main reasons why smaller companies are reluctant to trust AGVs, especially as they might not afford to have an engineer on site even during the day.

In those situations it would be very beneficial if any worker on site could reconfigure the path of the AGV. The purpose of this master thesis project is to solve this specific problem: how to enable a fast reconfiguration of the predefined path of AGVs for everyday users. This would benefit the smaller-sized companies and make AGVs a more accepted part of every company.

An increased use of AGVs in the industry has several sustainability benefits. Increased productivity, lower operating costs and decreased storage requirements are some of the economical ones. Also, AGVs need less space for maneuvering than manual forklifts, allowing tighter layouts in manufacturing. Another important aspect is the social benefits. Today one of the biggest causes of injuries at work are manual trucks[3]. An increased number of AGVs in exchange for manual forklifts will, most probably, reduce the number of injuries at work.

This project will also provide a link between research and industries. As is often the case the research today has come a long way further than industries, and projects like this are important to make sure that the industries keep evolving as well. The results from this project will also give feedback to the researchers, allowing them to confirm that their focus is still in line with that of the industries.

1.1 Objective

The aim of this project is to enable a more user-friendly way of reconfiguring the path of an AGV. This would allow a person inexperienced in programming to generate a change and to produce a driveable path.

1.2 Scope

This project will focus on extending the software used by AGVE in order to redesign the path of one individual vehicle at a time. No consideration will be given to the traffic controller and no adjustments to hardware will be made.

The vehicle this project will focus on is a three-wheeled forklift. The steering wheel is located in the middle of the front of the vehicle and two support wheels are located at the back of the vehicle.

The goal is to have a fast reconfiguration option in the user interface of the AGV. The implementation should allow any authorized user to change the path in a predictable and intuitive way. It should work both with and without a touch screen interface.

The project will only consider obstacles affecting the path between two consecutive nodes. It is assumed that the distance between the nodes is sufficient to create a driveable path around the obstacle. The new path can only be generated between nodes that can be reached by going forward from the current node, i.e. recording a path by reversing is not supported.

1.3 Work division

The authors of this report worked together during most of the project and helped each other as best as they could. The work was divided as follows: Martin had all the responsibility for programming the simulation program and the following translation to the actual software; Anna had the major responsibility for curve generation (performed in MATLAB) including adaption of curves. The rest of the work (literature search, report writing, test-drives and so on) was not divided, but performed in collaboration between the authors.

1.4 Outline

In chapter 1 the problem is introduced and the objective defined. The two following chapters, 2 and 3, describe the issues that will be addressed in the project as well as some previous work done in the area. Chapter 2 focuses on generating curves from given sample points while chapter 3 describes the AGV in more detail and gives some background on path generation.

Chapter 4 describes how knowledge from the previous chapters can be combined to achieve the main objective while chapter 5 presents a test scenario that is used to illustrate the results.

Chapter 6 evaluates the implementation and results, giving explanations to some choices made during the project. The chapter is complemented by chapter 7 where suggestions for further improvement of the final application are presented.

2

Translating sample data into continuous curves

In order for the AGV to be able to follow a path in a smooth and jerk-free way a continuous path is needed[4]. This makes it insufficient to create a new path by simply recording the position of the AGV. The samples need to be filtered (discussed in section 4.2.3) and interpolated (fitted) into a continuous curve.

This chapter describes how the sampled control points can be interpolated into a continuous curves. The chapter is divided into four parts: the first gives some background theory needed to understand the terms in the following sections, the second section outlines some of the previous research in the area, the third section focuses on curves relevant for this project while the last section presents some adaptions of the curves made in order to suit the main purpose of creating drivable paths.

2.1 Background theory

Terms commonly associated with curves relevant for this project are presented and explained in this section.

2.1.1 Parametric curves

There are two common methods of representing curves: implicit equations and parametric functions. Implicit equations describe the implicit relationship between x- and y-coordinates for all the points lying on the curve. Parametric functions[5], on the other hand, represent each coordinate separately, as an explicit function of an independent parameter.

Parametric curves have several advantages over implicit equations. Piegl and Tiller [5] state that “Parametric curves possess a natural direction of traversal . . . (making) it easy to generate ordered sequences of points along a parametric curve". Parametric curves are also better suited for representing shape in computer. Moreover, a parametric function provides more information than just a path: if the defining parameter is thought of as time, the function also gives the direction and speed of the particle as it moves along the path[6]. In general, a parametric function can cross itself or return to its starting position, which is impossible for implicit equations[7].

Given that the path generated in this project should be able to have an arbitrary shape and is to be implemented in a computer it is appropriate to use parametric curves in this project. Bézier curves, clothoids and B-splines are all examples of parametric curves.

2.1.2 Curvature

Curvature defines the change of a particle's heading with respect to distance traveled on the curve[8]. Higher curvature means that the segment is curvier than the one compared to. A circle has constant curvature which is inversely proportional to its radius.

Curvature is defined to be positive if the center is placed to the left when moving in the direction of increasing arclength, and negative otherwise. A spiral is defined as a curve with a monotonically increasing (or decreasing) curvature that never changes sign[9].

2.1.3 Hermite interpolation

Interpolation in this context means approximating sampled data to a function while fitting some distinct points exactly (interpolating). Commonly linear or quadratic polynomials are used to fit the function. Hermite interpolation takes the fitting a bit further: not only must some points be fitted exactly, but the derivative of the curve should be smooth too[10].

The purpose of Hermite interpolation is to generate curves well fitted to the sample data, while keeping the computational complexity low. One way of solving the problem is to use so called basis functions. Hermite basis functions have the following form:

$$\begin{aligned}
F_1(x) &= (x-1)^2(2x+1) \\
F_2(x) &= x^2(3-2x) \\
F_3(x) &= (x-1)^2 x \\
F_4(x) &= x^2(x-1)
\end{aligned} \tag{2.1}$$

Where x is between 0 and 1. Figure 2.1 illustrates the basis functions.

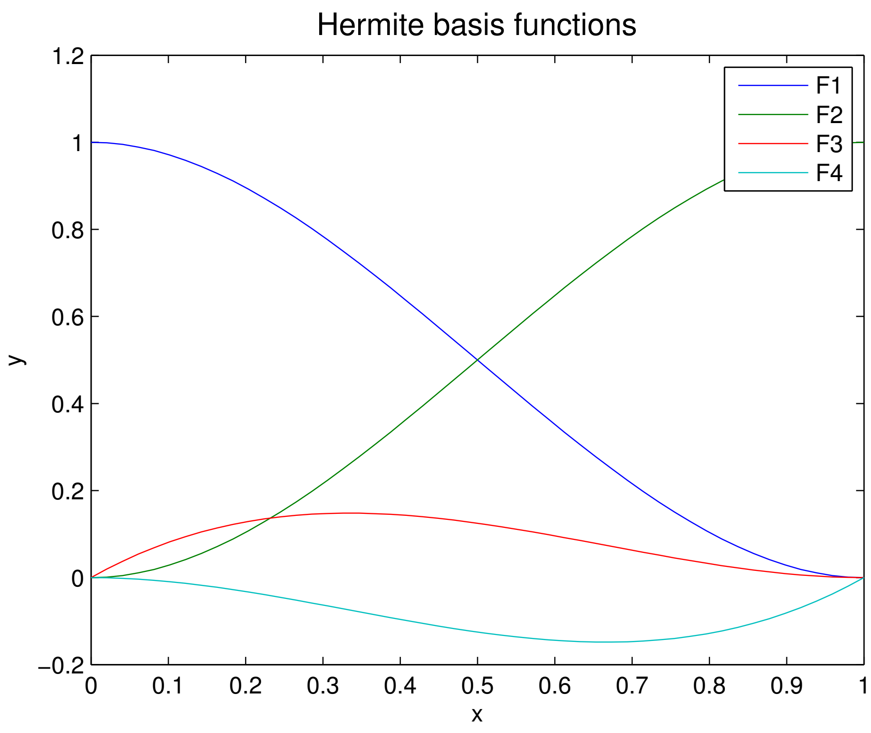


Figure 2.1: The Hermite basis functions.

The basis functions are blended to a final curve, $p(x)$, according to equation (2.2)[11]:

$$p(x) = y_0F_1(x) + y_1F_2(x) + m_0F_3(x) + m_1F_4(x) \tag{2.2}$$

Where y_i are the y-coordinates at point i and m_i is the slope of the polynomial at that point.

As equation (2.1) shows, the basis functions are cubic polynomials, which guarantees that two polynomials that meet at a point will have the same slope at this point. By fitting a basis function to every pair of sampling points a good approximation can be achieved while the polynomials are all of degree 3 or lower. This approach is called piecewise Hermite interpolation [11]. The piecewise Hermite interpolation function described here does not satisfy a global differentiability condition.

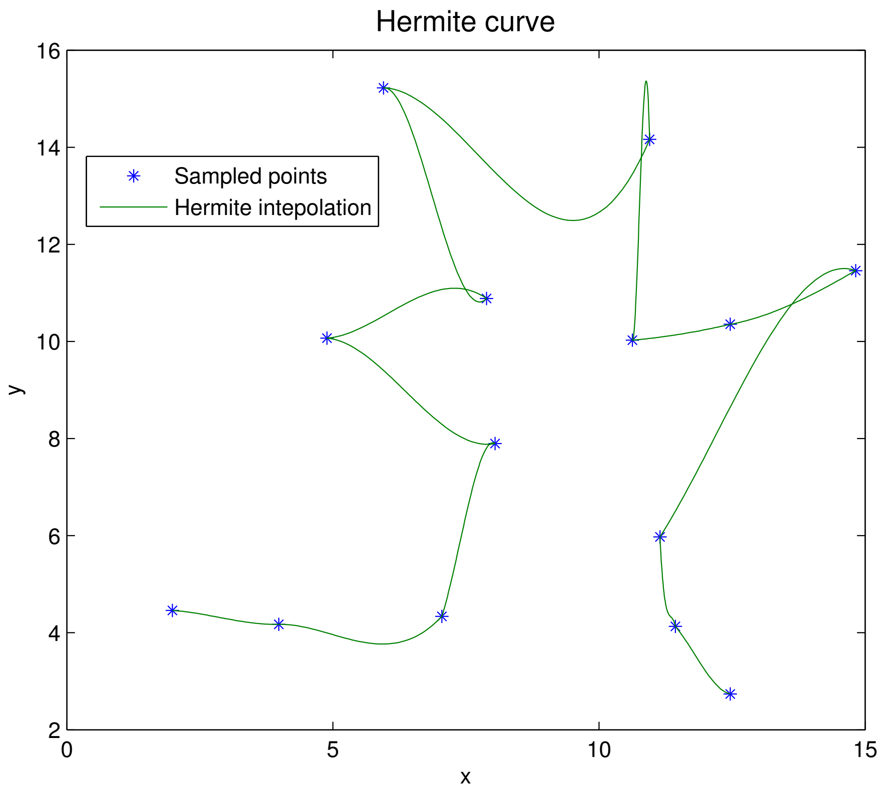


Figure 2.2: An example of implementation of Hermite interpolation in MATLAB. The curve looks differentiable as long as $x(i+1) > x(i)$ but if the relationship does not hold the smoothness of the shape is lost.

Figure 2.2 shows an example of implementation of Hermite interpolation curves in MATLAB. As the figure shows, the curve looks differentiable if $x(i+1) > x(i)$ but as soon as this relationship no longer holds the curve's shape will become unreasonable.

2.1.4 Continuity

A single function does usually not have enough freedom to represent a given curve in a satisfactory way. Therefore several segments are used instead, which need to be connected with some amount of continuity, to generate a curve of adequate smoothness[12]. There are several different kinds of continuity for parametric curves. Two of them will be mentioned here: the parametric and the geometric continuities.

Parametric continuity

Parametric continuity measures the smoothness of a curve by testing the continuity of its derivatives[12]. A curve with parametric continuity of degree k (denoted C^k) has a continuous k:th derivative. This means that any continuous/smooth function is of at least class C^0. A function of class C^1 is continuous and has a continuous derivative.

Depending on the purpose of the curve the required smoothness may differ. If, for instance, the curve is to guarantee that an object moves smoothly along it, both the path and the rate of change of path must be continuous. This is equivalent to stating that the curve must be C^1 continuous[6].

Geometric continuity

Geometric continuity is denoted as G^k, where k gives the increasing measure of smoothness. If two segments on either side of a point of a curve touch, they are of class G^0 . A curve of class G^1 ensures continuity of the tangent at the point. If the segments share a common center of curvature at the join point the continuity is of class G^2[13]. Any polygon can be interpreted as a control polygon of a G^k spline (splines are explained in section 2.1.5).

Geometric continuity is a less restrictive form of continuity than the parametric. Geometric functions are independent of parameterization, but still sufficient for giving the resulting curves geometric smoothness. They are especially appropriate for spline development as the extra freedom means that the design is more flexible than for parametric continuity[6]. If parametric continuity of order n exists, it implies that geometric continuity of order n does too. This does not apply the other way around, as geometric functions are a relaxed form of parametric continuity[12].

2.1.5 Splines

Initially, the term spline meant “a thin metal or wooden strip that is bent elastically so as to pass through certain points of constraint”[11]. Splines of that kind have been used for a long time in ship construction. According to the laws of physics the strip will acquire a form that minimizes the strain on it. This is a minimum energy problem that is difficult to solve directly, but can be well approximated by a cubic spline[11] (now ”spline” is used in its mathematical context).

A spline in the mathematical context is a function of degree m for which there exists $n+1$ knots, starting with a knot in the beginning of the function and ending at the end, such that between two knots the spline is a polynomial of degree $\leq m$[11]. Depending on the degree the splines are called linear (degree 1), quadratic (degree 2) or cubic (degree 3). Splines have continuity of degree C^{m-1}.

The above explanation can be reformulated as follows: splines are parametric piecewise functions where each piece (called segment) of the curve is a parametric function as well. The points where the segments join together are called knots and one of the key properties of the splines is that they are continuous at the knots[6]. Between two knots the splines usually have very simple form, but the total curve can be very complex and, most importantly, smooth.

2.2 Related research

”The NURBS Book” by Piegl and Tiller [5] presents numerous algorithms for generating NURBS (NonUniform Rational B-Spline), Bézier curves and other splines. Piegl and Tiller cover all aspects of splines, specifically of NURBS, necessary to design geometry in CAD, including ways of adapting the splines to ensure better interpolation of the sample data.

Another author who talks about splines and presents some algorithms related to those is Max Agoston. In his book ”Computer Graphics and Geometric Modeling”[11] a more detailed description of parametric curves in general can be found as well as some tips and ideas for curve-fitting. Agoston derives the splines in a more systematic way than Piegl and Tiller, starting with the origins of the word ”spline” and first tries of interpolating sample data and proceeding all the way to NURBS and clothoids.

Ho and Cook [14] show how third and fourth order spline functions can be used to generate trajectories for industrial manipulators. Their work was later extended by Petrniec and Kovačić [15] to include path generation suitable for AGVs. Petrniec and Kovačić’s approach is to generate continuous Bézier curves before applying the Ho and Cook algorithm between knot points given by quantizing the Bézier curve. The type of vehicle their simulations are done with are similar to the tricycle-type used at AGVE, however, real world application where their results have been used have not been found.

Another way of generating continuous curves is presented in the article “Sketching piecewise clothoid curves” by McCrae, J. and Singh, K[16]. Here a method for designing robot-vehicle path using sample data and fitting lines is presented. Circular arcs and clothoids produce an effective route using a stable and efficient algorithm[16]. The article also mentions NURBS, as a traditional way of designing curves.

Wang et al. [17] discuss a method to approximate clothoids by Bézier curves or B-splines to be able to use them in existing CAD software. In ”Handbook of Computer Aided Geometric Design” [13] several authors contribute to the discussion giving examples of different curves, techniques, models etc. for fitting curves to data.

Overall, the methods for fitting curves to sample data are many, and they are well researched. The challenge is to find methods relevant for this project, and adapt them to generate driveable AGV paths. The adaption of the curves to the relevant purpose is not as thoroughly investigated as the initial curve generation, meaning that methods for that will have to be developed during this project.

2.3 Curve theory

There exist many different algorithms for evaluation of sample data points and fitting them to appropriate curves. In this section several of the most established curves in the context of AGV routing are presented. Figure 2.3 illustrates two kinds of Bézier curves, NURBS and clothoids, compared to the sampled points.

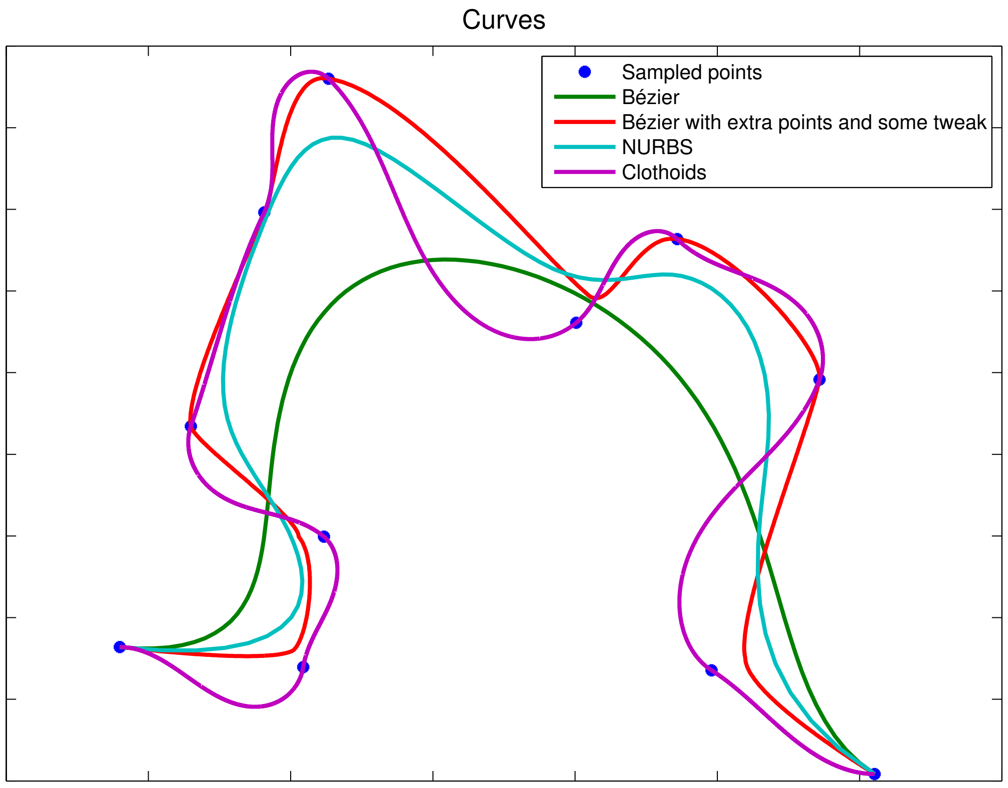


Figure 2.3: Different curves compared to sample data. The first Bézier curve is implemented using equation (2.4). Second Bézier curve, with tweaks, uses piecewise cubic curves. NURBS have knot vector $U = [0, 0, 0, 0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1, 1, 1, 1]$, with $p = 3$ and all control points have the weight 1. Clothoids start and end angles are approximated using the next sample point, expect for the last angle which is set to 0.

It can be seen from the figure that the different curves have different properties. A more detailed evaluation and comparison of the curves will be made in later chapters.

2.3.1 B-splines

A B-spline is an approximation of the spline exhibiting local control[18]. This means that moving a control point only has local effect on the curve, making it relatively easy to reshape the curve according to ones wishes.

"B" in B-spline stands for "basis" to emphasize that B-splines form the basis for the space of splines. In fact, any spline is linear combinations of B-spline basis functions[11]. The general form of a B-spline basis function of order k and degree $k-1$ is shown in equation 2.3:

Given k = 1

$$N_{i,1}(u) = \begin{cases} 1, & \text{for } u_i \leq u < u_{i+1} \\ 0, & \text{else} \end{cases}$$

If k >1

$$N_{i,k}(u) = \frac{u - u_i}{u_{i+k-1} - u_i} N_{i,k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} N_{i+1,k-1}(u) \tag{2.3}$$

u_i is a knot in the knot vector $U = [u_0, u_1, \ldots, u_n]$, where n is equal to the number of control points plus the degree of the curve plus one[19]. Each control point is associated to a knot. The number of control points associated to the same knot decides the multiplicity of a knot[11]. The smoothness of a knot depends on its multiplicity, with multiplicity of one corresponding to C^2 continuity, multiplicity of two to C^1 continuity and multiplicity of three to C^0 continuity. Note that the knots must be specified in non-decreasing order[20].

A knot vector can be either uniform or nonuniform (sometimes called periodic respectively nonperiodic). Uniform parameterization encourages the curve to spend approximately equal amount of "time" between each control point, meaning that if two control points are close to each other the shape will be forced to curve more (have larger curvature) than if the two points are far apart[7].

Depending on the order of the basis functions the B-spline's shape and smoothness will differ. A B-spline consisting of basis functions of degree 1 (order 2) is called linear B-spline and gives a continuous curve. A B-spline consisting of basis functions of degree 2 is called quadratic B-spline and one consisting of basis functions of degree 3 is called cubic B-spline. The higher the degree (corresponding to higher k in C^k) the smoother the transition between two basis functions will be, starting with C^0 for degree 1.

Every basis function exists only in a specified interval, which depends on the order of the function. Higher order means that the shape of the basis function is influenced by a greater number of points on either side of it. For instance a cubic B-spline basis function is influenced by two points on either side of the actual point of interest[11]. The B-spline is the sum of all basis functions existing at the relevant point.

Any segment of a B-spline curve of order k is contained in the convex hull of a corresponding sequence of k control points[11]. This can be problematic when designing a path for an AGV, as it might imply that the AGV will always end up inside the obstacle area. However, a B-spline curve can be forced to interpolate control points, meaning that the problem can be solved either by inserting multiple extra control points or by increasing the multiplicity of the knots. It should also be noted that when choosing parameters for the B-spline with a nonuniform knot vector the number of control points must be at least as large as the order of the spline[11].

A special type of B-spline curves are Bézier curves. Agoston [11] presents a theorem stating that each cubic B-spline curve can be thought of as a collection of cubic Bézier curves. In order to find Bézier control points corresponding to a B-spline of order k one simply has to insert knots until all have multiplicity $k - 1$.

2.3.2 Bézier curves

A Bézier curve can be defined by any number of control points, but the degree of the curve increases as the number of control points does. A Bézier curve of degree n is C^n

continuous and defined according to equation (2.4) [5]:

$$\boldsymbol{C}(u) = \sum_{i=0}^{n} B_{i,n}(u)\boldsymbol{P}_i \qquad 0 \leq u \leq 1 \tag{2.4}$$

where $\boldsymbol{C}(u) = \begin{bmatrix} x(u) \\ y(u) \end{bmatrix}$, $\boldsymbol{P}_i$ is a vector of control points, parameter u can be seen as a time parameter and $B_{i,n}(u)$ is the blending function specified in equation (2.5):

$$B_{i,n} = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i} \tag{2.5}$$

Creating curves using this approach ensures that the path starts at the first control point and ends in the last, but does not guarantee interpolation of the intermediate control points. Besides, change of one control point affects the shape of the entire curve. This makes it difficult to control the shape of the curve, making the approach not appropriate for this project.

Instead a solution presented in [11] was found suitable. Here the author suggests to use piecewise cubic Bézier curves. A cubic Bézier curve is a curve of degree 3, meaning that it has four control points. By introducing two new points between each of the given control points it is possible to force a cubic Bézier curve between every pair of control points, thus creating a curve consisting of several segments (pieces).

Using piecewise cubic Bézier curves results in a loss of the overall smoothness of the curve. This can be amended by introducing more points or by slightly moving the control points. The second approach is implemented in this project, giving the joint parts of the curve C^1 continuity. How the points are moved depends on their location. The purpose is to keep all of the curve outside the given control points and keep the edges smooth. Figure 2.3 shows an example of the implementation in MATLAB.

Bézier curves have two properties that are useful when designing AGV paths[21]. The first property defines that the distance between two relative points (those are found for different values of u) is relative to the current curvature. As AGVs usually have radius limitations, such as minimum turning radius or other undesired radii, it is important to be able to make accurate curvature calculations. The second property is that Bézier curves are tangent to the first and last segment at the beginning respectively end of the curve. This makes it easy to connect different path-segments and ensures G^1 continuity at the joints.

2.3.3 Rational splines

The functions discussed so far have one important disadvantage: there are some simple curves which they can not represent exactly[11]. In order to be able to represent forms such as circles, conic curves and ellipses a new class of curves needs to be introduced.

Hohmeyer et al. [12] state that "Any parametric polynomial curve can be expressed as a rational curve, and most polynomial splines and curves have rational extensions".

Their definition of a rational function, $r(u)$, is presented in equation (2.6):

$$r(u) = \frac{f(u)}{g(u)} \tag{2.6}$$

where f and g are polynomials in u.

Equation (2.6) can be rewritten according to equation (2.7), which is a more familiar form when discussing splines[11]:

$$r(u) = \frac{\sum_{i=0}^{n} w_i b_i(u) \boldsymbol{p}_i}{\sum_{i=0}^{n} w_i b_i(u)} \tag{2.7}$$

where b_i are appropriate basis or blending functions, w_i is the weight of control point i and $\boldsymbol{p}_i$ is the control point i.

If b_i are chosen as equation (2.5) and domain of u is $[0, 1]$ then equation (2.7) is called a rational Bézier curve. If b_i are chosen as equation (2.3), with a uniform knot vector, the curve $p(u)$ is called a rational B-spline curve of order k[11].

The weights must all be positive[20]. By varying the weights of different control points one can vary the shape of the curve. Increasing the weight will make the curve get closer to the corresponding control point, while decreasing the weight will mean decreased importance of the control point on the total shape of the curve. It should also be noted that ordinary Bézier and B-spline curves are special cases of the rational ones, when every point has been given an equal importance (i.e. by choosing $w_i = 1$ for all i)[11].

2.3.4 NURBS

NonUniform Rational B-Splines, NURBS, are a generalization of nonrational B-splines, as well as rational and nonrational Bézier curves. NURBS are very popular for defining curves in CAD/CAM systems. There are many reasons for that, among the most important are the facts that evaluation of algorithms is quite fast and computationally stable and that the NURBS allow a very flexible way of designing a large variety of shapes[19].

NURBS are expressed by inserting basis functions from equation (2.3) in expression for rational curves in equation (2.7). The term “NonUniform” means that the knot vector chosen for the spline is nonuniformly spaced. It is not trivial to choose an adequate knot vector. Piegl [19] suggests shaping an initial knot vector according to equation (2.8).

$$\boldsymbol{U} = [0, 0, \dots, 0, u_{p-1}, \dots, u_{n-p-1}, 1, 1, \dots, 1] \tag{2.8}$$

where p is the degree of the function (in equation (2.3) called $k - 1$) and $n = r + p + 1$ where r is the number of control points. The end knots are repeated $p + 1$ times and the number of the same knots in a row in the intermediate part of the vector should be less than the degree of the curve [20]. Here the knot vector is spaced between 0 and 1, but depending on implementation different values could be used.

2.3.5 Clothoids

Clothoids, also known as Euler spirals, spiros or Cornu spirals, are very common when designing railways and roads[22]. Their purpose is to make the transition from a circular arc to another circular arc or to a straight segment smooth and jerk free[9]. "Clothoid splines" is a term defining sequences consisting of lines, circular arcs and clothoid curves, thus having piecewise linear curvature profile[23].

A clothoid curve is defined in terms of Fresnel integrals, making it neither polynomial nor rational curve[17]. The Fresnel integrals are defined according to equation (2.9):

$$\begin{aligned} C(t) &= \int_0^t \cos(\frac{\pi}{2}\tau^2)d\tau \\ S(t) &= \int_0^t \sin(\frac{\pi}{2}\tau^2)d\tau \end{aligned} \tag{2.9}$$

Using equation (2.9), equation (2.10), describing the general parametric form of a clothoid spiral curve, can be derived. Bertolazzi and Frego [22] provide details about the derivation.

$$\begin{aligned} x(s) &= x_0 + \int_0^s \cos(\frac{1}{2}\kappa'\tau^2 + \kappa\tau + \theta_0)d\tau \\ y(s) &= y_0 + \int_0^s \sin(\frac{1}{2}\kappa'\tau^2 + \kappa\tau + \theta_0)d\tau \end{aligned} \tag{2.10}$$

where s is the arclenght, (x_0,y_0,θ_0) is the initial position including the start angle and $\frac{1}{2}\kappa' s^2 + s\tau + \theta_0$ is the angle at the final point[22]. Note that this implies that a clothoid only exist between two points, where the direction of the points needs to be known.

One of the properties of clothoids is that their curvature varies linearly with arclength, which in equation (2.10) is represented by $\kappa' s + \kappa$ [22]. This means that a vehicle following the curve will have a constant rate of angular acceleration. This property is very important when designing paths for AGVs as it makes it possible to enforce limitations on the turning radius, which is proportional to curvature, in an efficient way[24].

The fitting of clothoid curves is implemented using the method described by Bertolazzi and Frego[22]. Their approach reduces the G^1 Hermite interpolation problem to a single nonlinear equation that is then solved using the Newton-Raphson method. By providing a well chosen initial guess their approach finds a solution in 4-5 iterations.

This method produces a curve that is optimal with respect to the minimum curve length, as long as the difference between the start and end angles is in the interval $]-\frac{\pi}{2}, +\frac{\pi}{2}[$. It is possible to minimize the curve length for larger angle differences, but the results are unsatisfactory due to the fact that the solution becomes discontinuous -

a slight increase in angle may reverse the direction of the clothoid. For equations and algorithms the reader is referred to[22].

2.4 Curve adaption

In this section the previously defined curves are adapted to suit the objective of this project. Four kinds of curves are investigated: cubic B-splines, piecewise cubic Bézier curves, NURBS and clothoids.

2.4.1 Cubic B-splines and piecewise cubic Bézier curves

The previous section showed that there are many similarities between B-splines and Bézier curves. It is therefore appropriate to compare the two.

A piecewise cubic Bézier curve, as well as a cubic B-spline, are presented in figure 2.4. The figure shows that the Bézier curve is not smooth in some points and cuts a lot of corners. The cubic B-spline is smooth and interpolates the sample data quite well, but it too often cuts corners. Both would hence be problematic to use for generation of drivable AGV paths.

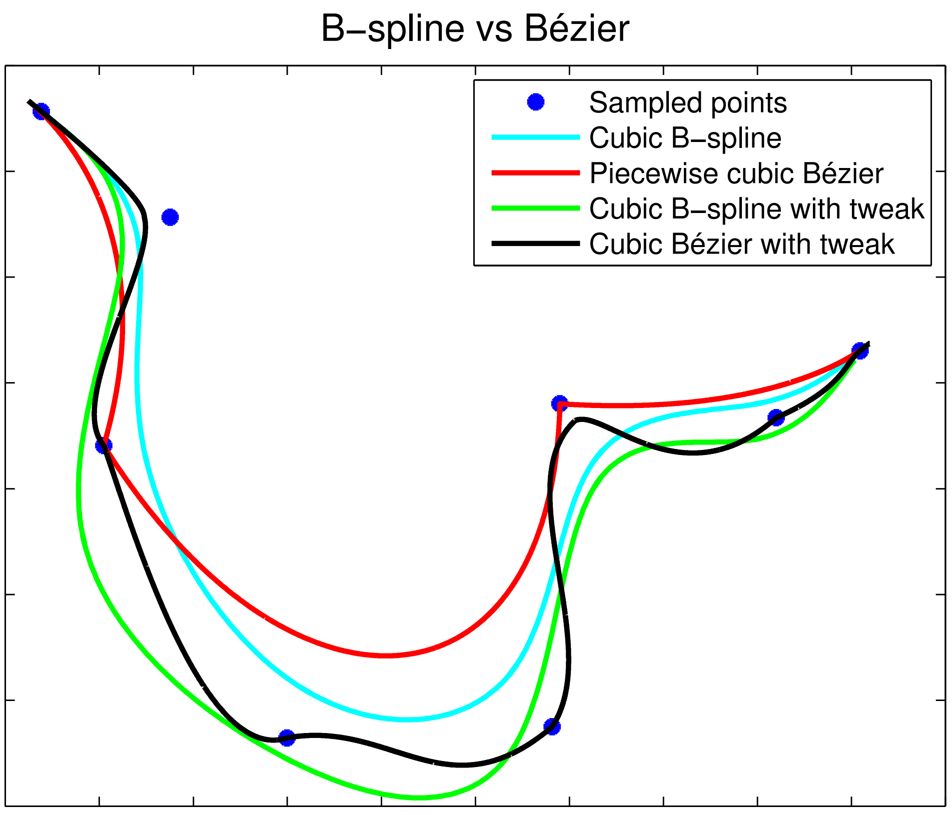


Figure 2.4: Comparison between cubic B-splines and piecewise cubic Bézier. Both types of curves are shown with and without adaptions.

Therefore the curves are tweaked. Bézier curve is tweaked using a set of rules describing each situation possible for the curve, such as ”upper left corner in an upward going path”. This process is slow and ineffective, as a large number of rules needs to be defined, but gives plenty of freedom for curve formation. If the character of the obstacle is known in advanced this freedom can be put to good use.

The cubic B-spline curve is tweaked by creating new points to produce a better fit. This process is presented in algorithm 1 and only deals with a few cases, thus not giving the freedom of Bézier curves, but is much faster and less parameter dependent.

$p^i_{check} = [x^i_{check}, y^i_{check}]$ in algorithm 1 is a point on the curve that corresponds to the given sample data point i. The given data points are labeled $p^i_s = [x^i_s, y^i_s]$, where s stands for *sampled*. p_{check} are found by splitting the curve into equally spaced segments, where each segment has the same number of points in it, points in this case being the ones that build up the curve, not p_s. This is possible to do as the uniformly spaced knot vector guarantees that the same number of points is located between every p^i_s[1]. The newly created sample points, used for calculation of the tweaked curve, are labeled p_{as}. See figure 2.4 for comparison between the tweaked curves and the initial ones.

Algorithm 1 Tweaking of cubic B-spline curves

```
1: while $p^i_{check}$ is on the inside of $p^i_s$ (inside the obstacle area) do
2:     if $x^i_{check} \geq x^i_s$ then
3:         Move $p^i_{as}$ to the right
4:         Move $p^i_{as}$ up or down, depending on the form of the curve
5:     else
6:         Move $p^i_{as}$ to the left
7:         Move $p^i_{as}$ up or down, depending on the form of the curve
8:     end if
9:     Calculate the new curve
10:    Calculate the new $p^i_{check}$
11: end while
```

2.4.2 NURBS

There are several ways of adapting the NURBS curves to fit the sample data in a way more adequate for path generation. Two different scenarios can be studied: the sample points are either many and closely spaced or few with large distances in between. As both Bézier curves and B-splines discussed in section 2.4.1 are adapted to few sample points only the scenario with many data points is studied for NURBS.

One of the first issues to consider when adapting NURBS is the knot vector. N in NURBS suggests that the knot vector should be nonuniformly spaced, even though it is common to ignore that fact. Piegl and Tiller [5] present two ways of spacing the knot

[1]Note that the found points do not correspond to the sample points exactly. The exact match can be found by taking the average of several knots for every sample point, while weighting them differently depending on the degree of the curve[5]. To make a simple division was found sufficient in this context.

vector. The first one is called the chord length method and spaces the knot vector based on the length between the sample points. According to the authors this method is the most common. The second one is called the centripetal method and is the newer one of the two. It is supposed to work better when dealing with sharp corners. It uses the square root of the distance between two sample points, whereas the chord length method uses the distance it self, see equation (2.11). P_k is the k:th sample point, n is the number of sample points (starting from 0), p is the degree of the function and u_i is an element in the final knot vector.

$$
\begin{aligned}
d &= \sum_{k=1}^{n} \sqrt{|P_k - P_{k-1}|} \\
\overline{u}_0 &= 0 \qquad \overline{u}_n = 1 \\
\overline{u}_k &= \overline{u}_{k-1} + \frac{\sqrt{|P_k - P_{k-1}|}}{d} \qquad k = 1,\ldots,n-1 \\
u_0 &= \ldots = u_p = 0 \qquad u_{m-p} = \ldots = u_m = 1 \\
u_{j+p} &= \frac{1}{p} \sum_{i=j}^{j+p-1} \overline{u}_i \qquad j = 1,\ldots,n-p
\end{aligned}
\tag{2.11}
$$

Both methods were implemented in MATLAB and results showed that the curves with nonuniform knot vector were smoother than the one with uniform knot vector. The centripetal method did give less sharp corners therefore it was chosen for future implementations. Neither of the methods did by itself enable good interpolation of the given data points.

If many sample points are available the NURBS curve will, even without any adjustments, follow the sample data well. Even if the centripetal method is used the curve may still have sharp corners, which is not acceptable when designing an AGV's path. One method of dealing with sharp corners while keeping the path outside the obstacle is described in algorithm 2, where the sample points, $[xs, ys]$, are "resampled" (similarly to how the B-splines were in section 2.4.1). The method is intended to work when many sample points are available and makes the corners less sharp, see figure 2.5. There are other ways of adapting the NURBS curves to fit ones purpose, such as knots insertion, weight adjusting and curve splitting (see [5] for more information about those) but they are not studied further in this project.

Algorithm 2 Dealing with corners on NURBS curves

```
1: for all $i$ do
2:     if $P(i) = [x(i) \quad y(i)]$ is the point at the peak of the corner then
3:         Place a point $p$ halfway between $P(i-2)$ and $P(i+2)$
4:         Find the slope coefficient, $k$, of a line, $y_p$, between point $p$ and the peak $P(i)$
5:         Find lines, $y_1$ and $y_2$, parallel to $y_p$, through the points $P(i-2)$ and $P(i+2)$
6:         $n \leftarrow \frac{-1}{k}$
7:         $m \leftarrow y(i) - n * x(i)$
8:         $y \leftarrow n * x + m$ is the equation stating the level of the peak
9:         Find the points where $y$ intersects with $y_1$ and $y_2$ respectively
10:        Move $P(i-1)$ and $P(i+1)$ to the points of intersection
11:    end if
12: end for
```

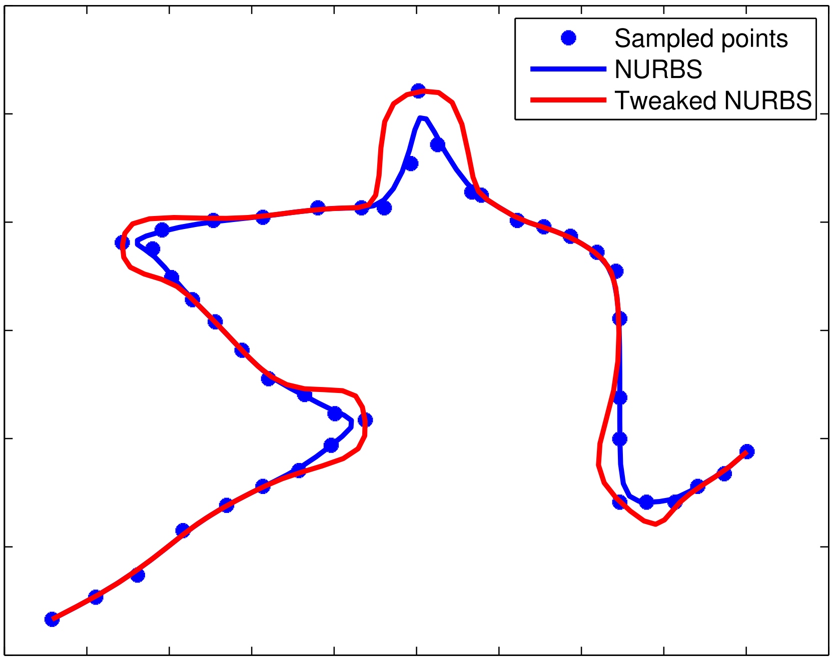


Figure 2.5: Tweaking of a NURBS curve with many data points available. By moving the two adjacent points level with the corner's peak less sharp corners are achieved. The weights are all set to 1, the knot vector is spaced using the centripetal method and the degree is equal to 4.

2.4.3 Clothoids

Because a clothoid is only defined between two points, a curve consisting of several sample points will include a number of clothoids. A curve consisting of several clothoids can be seen in figure 2.3. As clothoids are relatively computationally heavy, it is, however, not common to use only clothoids to create a curve. Instead a combination of straight lines and circular arcs, connected by clothoids, is preferred (see for example [16], [24] and [23]).

A method for creating curves using a combination of straight lines and clothoids is presented in algorithm 3. It aims to find as many long lines in the sample data as possible and connect those using clothoids. This method greatly reduces the number of clothoids. The following parameters are used in the algorithm: *ShortestLine* defines the shortest allowed length for a line, lines shorter than that are simply not checked; *Distance* provides the shortest allowed distance between two lines, one that is sufficient to insert a clothoid that makes the transition between the lines smooth; *Tolerance* defines the largest allowed distance between a point along the curve and the corresponding point on the suggested line; *AreaTolerance* makes sure that the line is centered between the points; *DoneWithLine* is a parameter used to check the possibility of creating a line from the current starting point i and making it longer; *LineLength* is the length of the line currently evaluated.

Algorithm 3 Approximating the sampled path using long lines and clothoids

```
1: for all samples do
2:     $LineLength \leftarrow ShortestLine - 1$
3:     $DoneWithLine \leftarrow$ false
4:     while not $DoneWithLine$ do
5:         Create a line from $i$ to $i + LineLength$
6:         $Area \leftarrow$ total area above the line - total area below the line
7:         for every sample between the start and end of the line do
8:             $Distances \leftarrow$ the distance from the sample to the line
9:         end for
10:        if all $Distances < Tolerance$ && $Area < AreaTolerance$ then
11:            $LineLength \leftarrow LineLength + 1$
12:        else
13:            if $LineLength \geq ShortestLine$ then
14:                Save the line
15:            end if
16:            $DoneWithLine \leftarrow$ true
17:        end if
18:    end while
19: end for
20: while lines crossing other lines still exist do
21:    Find the longest line $j$
22:    Remove all lines that start or end inside it
23:    Remove $j$ from the list of remaining lines
24: end while
25: Between every two lines create a clothoid connecting them
26: If clothoids are needed to fill out the beginning or the end of the curve, create those
```

This method is suitable when there are many sample points available. Figure 2.6 shows a comparison between the sampled path and a curve built up of lines connected by clothoids. The shape of the path is well preserved, while (probably unintentional) dents are smoothed out and several long straight segments are achieved.

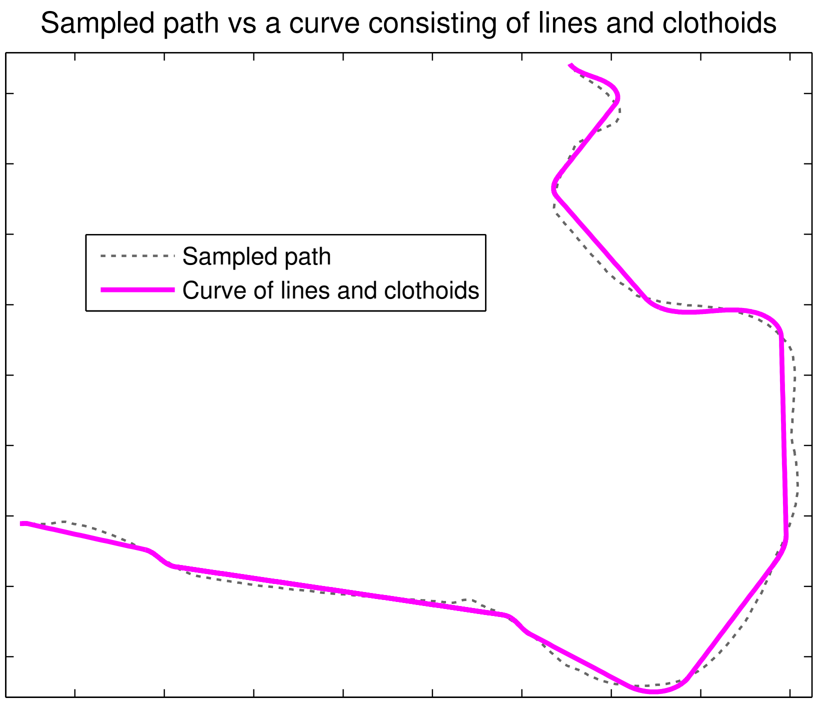


Figure 2.6: The sampled path compared to an approximating curve consisting of a combination of clothoids and lines. The initial form of the path is preserved, while several, probably unintentional, dents are smoothed out.

2.4.4 Comparison of the adapted curves

Figure 2.7 shows a comparison between the different curves generated from a set of sample data. The automatically sampled data is used for generating a NURBS curve and a curve consisting of lines and clothoids, while the manually sampled data is used to produce a B-spline curve, a Bézier curve and a curve consisting only of clothoids. Section 4.2.2 explains the difference between the automatically and manually sampled data.

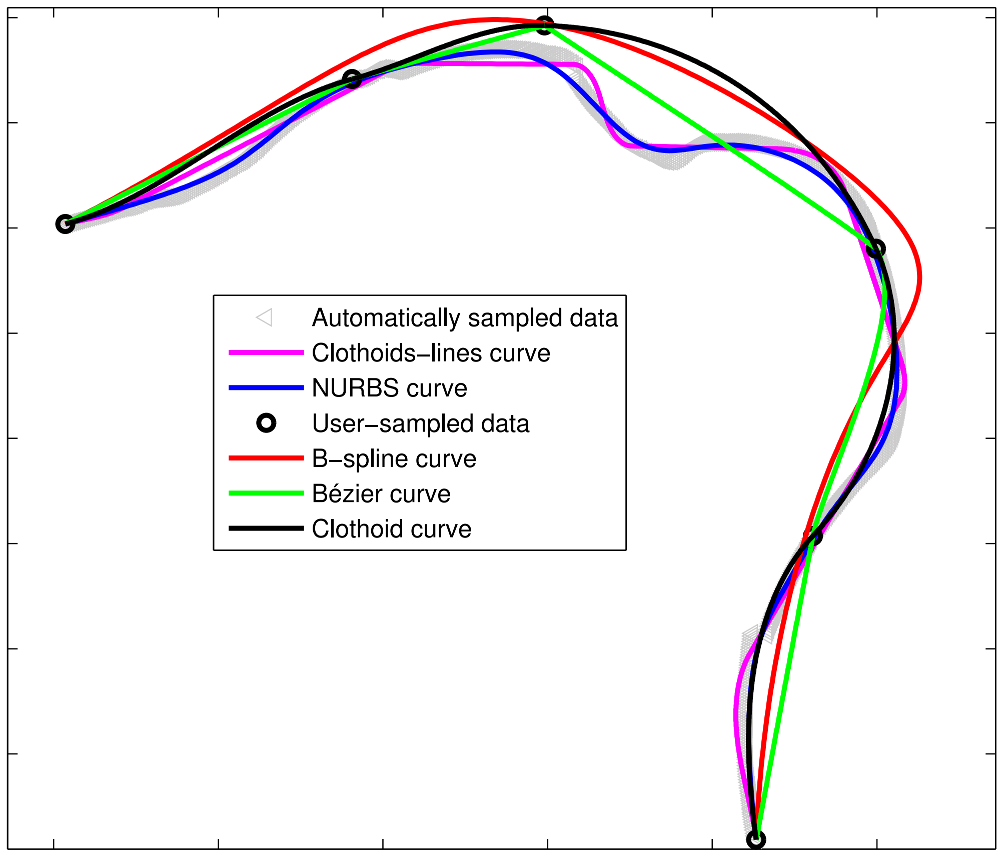


Figure 2.7: A comparison of different curves. The clothoids-lines curve and the NURBS curve interpolate the automatically sampled data, while the B-spline curve, the Bézier curve and clothoid curve fit the data sampled by the operator. Bézier curves are not included in the final application, reasons for this are explained in section 6.2.

Both the NURBS curve and the curve consisting of a combination of lines and clothoids follow the sample data well, even though the clothoids-lines curve approximates some parts of the curve with straight lines.

There are some pronounced differences between the three curves interpolating the manually sampled data. The B-spline succeeds in keeping the entire path outside the sampled points, while the curve consisting of only clothoids interpolates the data exactly. The Bézier curve also interpolates the data well, but it can be seen from figure 2.7 that not all sharp corners are removed. This implies that some scenarios were not included in the tweaking (see section 2.4.1). The inability to make sure that all the relevant scenarios are considered is the main reason for not including the Bézier curve in the final application. More reasons are stated in section 6.2.

3

AGV path planning

AGVs were first introduced in 1955 and are today common in many different areas of applications[1]. Application relevant for this thesis is AGVs used for loading, unloading and transporting materials in different environments. When planning the path of such a vehicle there are several issues to consider, such as the risk of tipping over and maintaining the overall flow in the facility.

This project focuses on designing a driveable path around an obstacle for a tricycle-type AGV. The model of the vehicle, as well as speed and curvature limitations, are presented in this chapter. In addition some of the related research in the area is covered.

3.1 Related research

Dahari and Yang present a review over different routing algorithms for AGVs[21]. They cover both wire-guided and free-ranging routing algorithms, where the later include Bézier curves and potential field navigation. Depending on the complexity of the layout, the needed accuracy and the available computational power, different algorithms perform the best. Overall, they recommend hybrid methods for both research and industrial applications.

Brezak and Petrovic are concerned with designing a path that is not only obstacle free but also feasible (in the sense that it is driveable) and optimal in the sense of smoothness[24]. Starting from an already existing path they presume to smooth it out using clothoid-segments. The algorithm they propose ensures G^2 continuity and low computational complexity.

In an article from 1995 Bissé et al. present algorithm to solve the inverse kinematics of a non holonomic system such as a tricycle AGV[25]. The tricycle in their example has the front wheel as the steering wheel and is driven by the two rear wheels. They also give an algorithm to iteratively find the best path between two points, while not taking obstacles into consideration, using parametric polynomial interpolation.

A steering method for car-like vehicles is presented by Fraichard and Scheuer[26]. Here too the algorithm plans the path while not considering the obstacles. The difference compared to Bissé et al. is that the entire path is planned, not just between two data points, and that curvature limitations are enforced on the path. Enforcing curvature limitations enables the car-like vehicle, the authors state, to follow the path without ever having to stop in order to reorient the front wheels. The path they suggest is made up from lines, arcs and clothoid segments.

Montés and Mora also show interest in paths consisting of clothoidal segments, but suggest a way of decreasing the computational complexity[27]. They propose to approximate the Fresnel integrals by Rational Bézier Curves and present an algorithm accomplishing that. This way, they claim, on-line path planning with clothoids can be achieved.

The process of defining a path and driving it is very well researched, both for different types of vehicles and different constraints. The twist in this project is that the obstacle the path should avoid is not known in advance. It is therefore appropriate to study different kinds of curves. In addition, the existing software requires the AGV to have a certain heading when following paths in the original layout. This means that the restrictions enforced on the newly generated path are harder than the ones suggested in most of the related articles.

3.2 AGV model

This project focuses on a tricycle-type AGV steered and driven by the front wheel placed along the center of the vehicle. The two rear wheels are support wheels. The wheelbase, wb, and the wheelwidth, ww, depend on the vehicle. Figure 3.1 shows a sketch of the AGV. In the figure both the steering angle, α, and heading angle, θ, are marked. Note that the steering angle is defined relative to the heading, while the heading is defined relative to the layout coordinate system.

The AGV is modeled as a rigid body moving on a plane. Its reference point P is placed at the front wheel (see figure 3.1). In the project no consideration to whether the truck is loaded and how high the fork is lifted, is taken. The AGV is non holonomic because it has constraints on turning radius and can only move in the direction perpendicular to it's rear wheel axle.

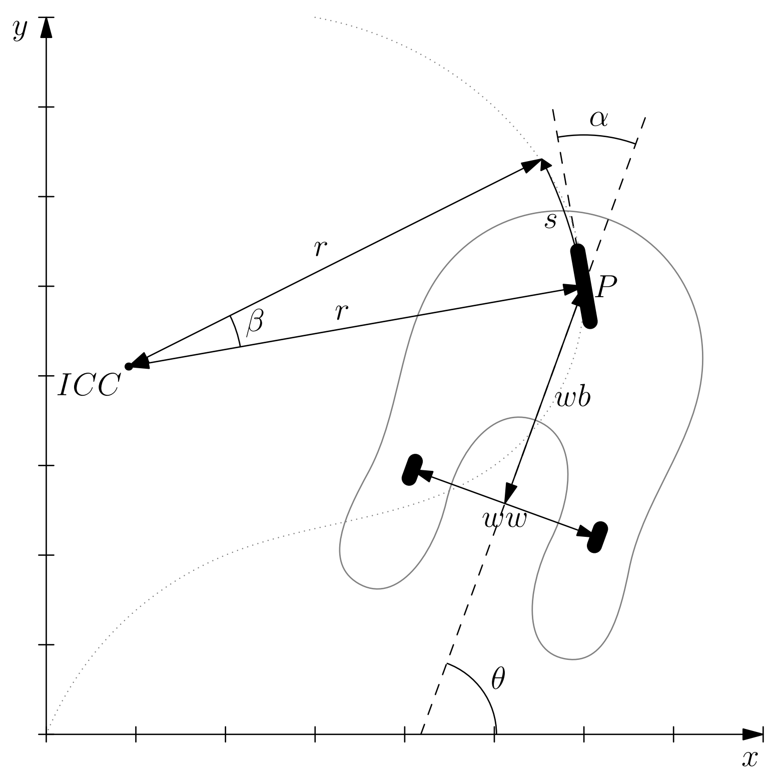


Figure 3.1: Model of an AGV. ICC is the Instantaneous Center of Curvature, α is the steering angle, θ is the heading angle of the AGV, r is the radius of the curvature at point P, wb is the wheelbase, ww is the wheelwidth, s is the segment length on the path corresponding to the steering angle α, β is $\Delta\theta$ meaning the difference in the heading of the AGV before and after the move and P is the reference point of the vehicle, located at the steering wheel.

The equations of movement of the reference point P=(x,y) can be derived using figure 3.1. The configuration space of the AGV is indicated by equation (3.1). v used in this equation is the speed of the vehicle along the path.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos(\theta + \alpha) \\ \sin(\theta + \alpha) \\ \frac{\sin\alpha}{wb} \end{pmatrix} v \tag{3.1}$$

Equation (3.1) can be discretized in terms of the segment length, s. This results in the nonlinear discrete state space model presented in equation (3.2). The states are x,y and θ and both α (the steering angle) and s are inputs to the system.

$$\begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} x(k) + \cos(\theta(k) + \alpha(k))s(k) \\ y(k) + \sin(\theta(k) + \alpha(k))s(k) \\ \theta(k) + \frac{\sin(\alpha(k))}{wb}s(k) \end{bmatrix} \tag{3.2}$$

Formulating the state space with the segment length as the step size, rather than time, results in slightly reduced accuracy. It is, however, convenient when simulating an AGV path, as the number of calculations is reduced, given that the segments are longer than $timestep * v$.

3.2.1 Path limitations

When designing a path it is important to consider how and if the AGV will be able to drive along it. Some of the relevant limitations are described in this section.

Condition for smooth motion

As long as the vehicle is in motion, the steering wheel is assumed to be aligned with the tangent direction of the path. This fact, combined with the placement of the reference point at the front wheel of the AGV, makes G^1 continuity of the path a condition for smooth motion[4]. Note that even though the steering wheel is aligned with the tangent direction of the path it is not guaranteed that the rest of the AGV is, as it takes time to straighten up the vehicle after a turn.

Steering actuator limitations

The steering wheel is limited to turn angles between $[-\frac{\pi}{2}, \frac{\pi}{2}]$, relative to the heading of the vehicle. Therefore it is easy to see that the wheelbase is limiting the turning radius of the AGV. This does not, however, imply that the path can not make turns that have a radius less than the wheelbase. Depending on the AGV's heading and the radius of the curve, the turn the AGV can manage differs.

It is important to predict the behavior of the steering angle, in order to ensure feasibility of the path. In this project the feasibility is checked using a simulation procedure described in detail in section 4.1.5. Another possible solution is to derive a mathematical relationship, relating the heading of the AGV and the shape of the curve to the needed steering angle. This is discussed in section 7.1.

The actuator for the steering wheel can not change direction instantaneously. Its maximum turning speed, $\dot{\alpha}_{max}$, is a parameter dependent on the vehicle. In AGVE's software the parameter is given in $[\frac{degrees}{s}]$ and should be taken in consideration when designing a new path.

The relation between speed of the vehicle, v, angle change compared to the previous segment, $\Delta\alpha$, the time it should take to complete the segment, Δt, the length of a segment, s, the and the actuator's turning speed $\dot{\alpha}$ are presented in equations 3.3.

$$\begin{aligned} \Delta t &= \frac{s}{v} \\ \dot{\alpha} &\approx \frac{\Delta\alpha}{\Delta t} \end{aligned} \tag{3.3}$$

Note that $\frac{\dot{\alpha}}{v}$ is the same as the curvature of the path, as it is equivalent to the change of direction over the distance travelled.

Avoiding unnecessary wear

Another issue to consider when designing a path is when the AGV is turning according to figure 3.2. This results in a scenario where the inner rear wheel stays fixed during the turn, causing extra wear on the wheel and the floor. It is thus desirable to avoid turns resulting in this turning radius.

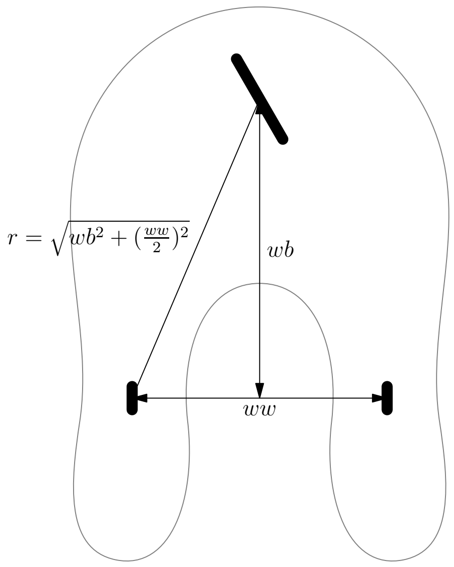


Figure 3.2: The equation for the undesirable turning radius, r, causing increased wear on the rear wheel and the floor, is shown in the figure. wb and ww are defined as before: wheelbase and wheelwidth respectively.

Tilt risk

The tilt risk is important to consider when designing a path for an AGV. In order to make accurate calculations in that area it is necessary to know the center of weight of the vehicle. AGVE has today no sensors to achieve that, therefore no calculations of that kind can be made. Instead fixed speed limitations are enforced.

When the fork is raised the vehicle's max speed is set to a predefined constant. Another limitation, based on the turning radius of the path, r, is shown in equation (3.4).

$$v = \sqrt{\frac{r}{K}} \tag{3.4}$$

This formula is used by AGVE when deciding the maximum speed, v. K is a design constant set to 0.015, based on experience of the engineers at AGVE.

4

Software: analysis and implementation

In this chapter the extensions made to AGVE's software are presented. They include modifications of the control laws, methods for generating the dynamic turns on-line and the user interface, created to facilitate the generation and removal of paths by an operator. Additionally the chapter explains and motivates different use cases supported by the final application and describes the flow of the program.

4.1 Adding dynamic turns to the existing software

This section discusses two kinds of turns: programmed and dynamic. The *programmed turns* are those that are created in the CAD-software and predefined by AGVE, while the *dynamic turns* are those generated when the user is redefining the path according to this project's problem formulation. The section describes how the dynamic turns are stored, how the position of the AGV is controlled when following a turn and how the dynamic turns interact with the existing software. A process to check the feasibility of the path is also presented.

4.1.1 Development environment

The AGV's control board is called CB80 and contains an Intel ATOM processor that runs a real time Windows CE operating system. The software is written in C++ and can be transferred to the AGV via USB or Ethernet.

4.1.2 The ACE interpreter and layout

All actions performed by the AGV are stored in binary tables. These contain instructions corresponding to commands such as "*move forward a certain distance*" or "*run* *programmed turn #3*", which are decoded by the ACE interpreter. This system has been used since AGVE was founded and computing power was a lot scarcer than it is today. As such, the software control of the AGV is intimately tied to the ACE interpreter.

ACE commands are sent from the controlling host (in production the host is the central traffic controller) and stored in a FIFO buffer on the AGV. The AGV pulls commands from this buffer and executes them one by one until the buffer is empty.

In addition to the ACE tables there exists a newer layout database containing the paths between the nodes and the tables they correspond to, as well as maximum initial speeds. The initial speeds are important because ACE makes sure to slow down to the given initial speed of the next segment (if there is one) before completing the current table.

As AGVE would like to move away from the old system with ACE tables in the future, the dynamic turns are implemented separately from the existing ACE infrastructure. The layout database provides a good starting point for this approach, as all the relevant parameters (such as speed and turn length) can be stored there as long as a dynamic turn is active. This allows the dynamic turn to intercept the table command corresponding to the segment in question while maintaining a smooth transition from/to other ACE commands.

4.1.3 Analysis of the programmed turns

The turns in the layout are implemented in a similar fashion as the ACE tables: they are short programs containing instructions for setting movement speed and steering angle (see listing 4.1 for an example). These turns are generated within CAD software, which makes them hard to change on-line. The actual algorithms used to design the turns are not known.

```
        PTURN 11
        SPEED 25
        GUIDANCE
        ST_ANG 0        ; Start angle
        EX_ANG 0        ; Exit direction
        SEGLEN 4.657    ; Segment length
        REPEAT 7, 116
        REPEAT 0, 95
        REPEAT −7, 116
        REPEAT 0, 3
        REPEAT −7, 116
        REPEAT 0, 95
        REPEAT 7, 116
;       X       Y       Angle
LOCATIONS:
        165926  93374   180.000
ENDLOC
```

```
EOP                    ; End of Pturn #11
```

Listing 4.1: Example of a programmed turn.

The `REPEAT` instruction increments the steering angle by a fixed number of degrees every timestep, which suggests that the programmed turns are in fact a combination of clothoids. In theory the vehicle will move in a circular arc around its instantaneous center of curvature, depending on the steering angle, α, and the heading of the AGV, θ (see figure 3.1). Equation (3.2) implies that over the course of the path the AGV will move according to algorithm 4. Note that *segmentLength* in algorithm 4 is equivalent to $s(k)$ in equation (3.2).

Algorithm 4 Theoretical AGV position when following a programmed turn.

```
1: while path not complete do
2:     $\alpha \leftarrow \alpha + steerIncrement$
3:     $r \leftarrow wb/\sin(\alpha)$
4:     $\beta \leftarrow segmentLength/r$
5:     $\theta \leftarrow \theta + \beta$
6:     $x \leftarrow x + segmentLength * \cos(\theta + \alpha)$
7:     $y \leftarrow y + segmentLength * \sin(\theta + \alpha)$
8: end while
```

Due to inaccuracies in the model and real world factors such as actuator speed, wheelslip, etc the AGV will drift slightly from the path if no feedback is given. On a modern AGV equipped with a laser scanner it is possible to use a PD controller to keep the AGV on track. For this to work the programmed turn is translated into discrete points in layout space according to algorithm 4. The deviation is then calculated by measuring the distance from the AGV to the path created by the previous and next point on the turn and correcting it with the distance from this path to the current point on the turn. The calculation of the deviation from the path is shown in equation (4.1) and the corresponding parameters are seen in figure 4.1. $sign_z$ in the equation stands for the sign of element z in the vector.

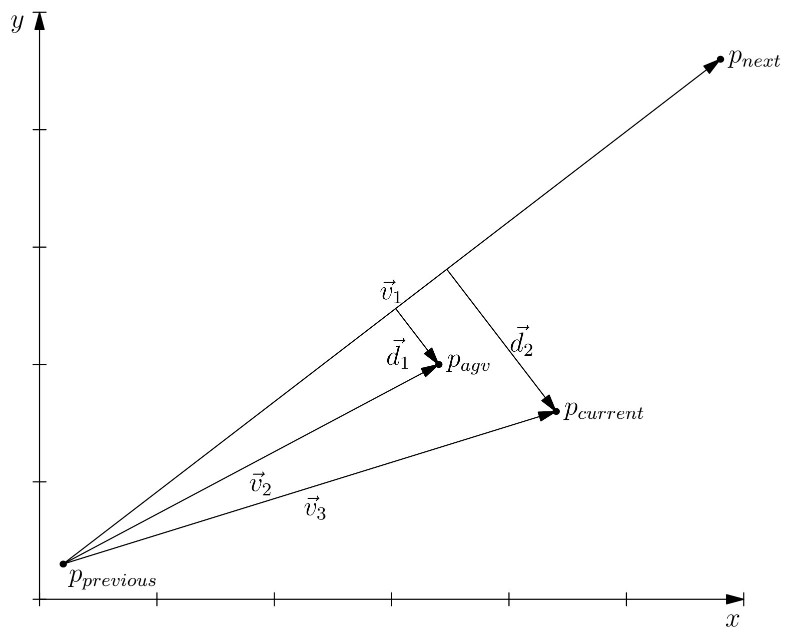


Figure 4.1: Calculation of deviation from path. $p_{previous}$, p_{next} and $p_{current}$ are points on the turn, p_{agv} is the actual position of AGV and $\vec{v}$ and $\vec{d}$ are distance vectors.

$$
\begin{aligned}
\vec{v}_{1u} &= \frac{\vec{v}_1}{||\vec{v}_1||} \\
\vec{d}_1 &= \vec{v}_2 - (\vec{v}_2 \cdot \vec{v}_{1u})\vec{v}_{1u} \\
s_1 &= sign_z(\vec{d}_1 \times \vec{v}_{1u}) \\
\vec{d}_2 &= \vec{v}_3 - (\vec{v}_3 \cdot \vec{v}_{1u})\vec{v}_{1u} \\
s_2 &= sign_z(\vec{d}_2 \times \vec{v}_{1u}) \\
d_{tot} &= s_1||\vec{d}_1|| - s_2||\vec{d}_2||
\end{aligned}
\tag{4.1}
$$

The steering angle is corrected according to $\alpha = \alpha + PD(d_{tot})$, where $PD(d_{tot})$ represents the PD controller used to regulate the deviation from the path d_{tot}. Each timestep the current point is updated, if the AGV has travelled far enough compared to the length of the current segment. This means that even though the controller works well, it can not handle drifting too far off course, as moving in the wrong direction would still move the current point forward. This is not a problem in practice as long as the paths generated adhere to the limitations of turning radius.

4.1.4 Adaptation of control law

In contrast to the programmed turn described in section 4.1.3, the dynamic turn is based on a mathematical curve rather than a stepwise incrementation of the steering angle. However, as both of them can be seen as a set of ordered points along the path the principle of control remains largely the same. In fact, the only difference is that the steering angle is directly set to align with the tangent direction of the curve at the current point on the path, rather than being continuously incremented. Because the curve and the AGV are in the same coordinate system, the following relation will give the correct steering angle: $\alpha = \phi - \theta$, where ϕ is the tangent direction of the curve. This is then corrected with the same PD controller as described in section 4.1.3. The resulting control law is not more computationally demanding than the original one, which implies that the time constraints enforced by the real time nature of the system are most likely satisfied. This is important to ensure that safety requirements are not compromised.

To correctly set the steering angle α the tangent direction of every point needs to be known. Another requirement is that the length between the points on the curve has to be defined, in order to calculate how far along the path the AGV has driven. The dynamic turns are stored as a list of points described by listing 4.2.

```
struct DT_POINT = {
  double x, y;
  double angle;
  double segment_length;
};
```

Listing 4.2: Dynamic turn point structure.

4.1.5 Simulating the AGV's movement

It is useful to be able to simulate the AGV's movement along a path on-line. Modifying algorithm 4 with the adaptations described in section 4.1.4 results in a slightly different algorithm for calculating the theoretical AGV position. In addition, as the simulation will be used to determine the feasibility of the path, it also needs to provide an estimate of the steering wheel's angular velocity, $\dot{\alpha}$ (see equation 3.3). The simulation procedure is presented in algorithm 5, where ϕ is the tangent direction of the current point and v is the forward speed of the AGV.

Algorithm 5 Theoretical AGV position and heading when following a dynamic turn

```
1: while path not complete do
2:     $\alpha_{old} \leftarrow \alpha$
3:     $\alpha \leftarrow \phi - \theta$
4:     Normalize $\alpha$ to be within $[-\pi,\pi]$
5:     Saturate $\alpha$ to be within $[-\pi/2, \pi/2]$
6:     $r \leftarrow wheelbase/\sin(\alpha)$
7:     $\beta \leftarrow segmentLength/r$
8:     $\theta \leftarrow \theta + \beta$
9:     $x \leftarrow x + segmentLength * \cos(\theta + \alpha)$
10:    $y \leftarrow y + segmentLength * \sin(\theta + \alpha)$
11:    $\dot{\alpha} \leftarrow v(\alpha - \alpha_{old})/segmentLength$
12: end while
```

By inspecting algorithm 5 it becomes clear that the AGV will correctly follow a given curve only if α never saturates and $|\dot{\alpha}|$ is never greater than the steering wheel's maximum angular velocity. If either of these situations occur the turn is not feasible.

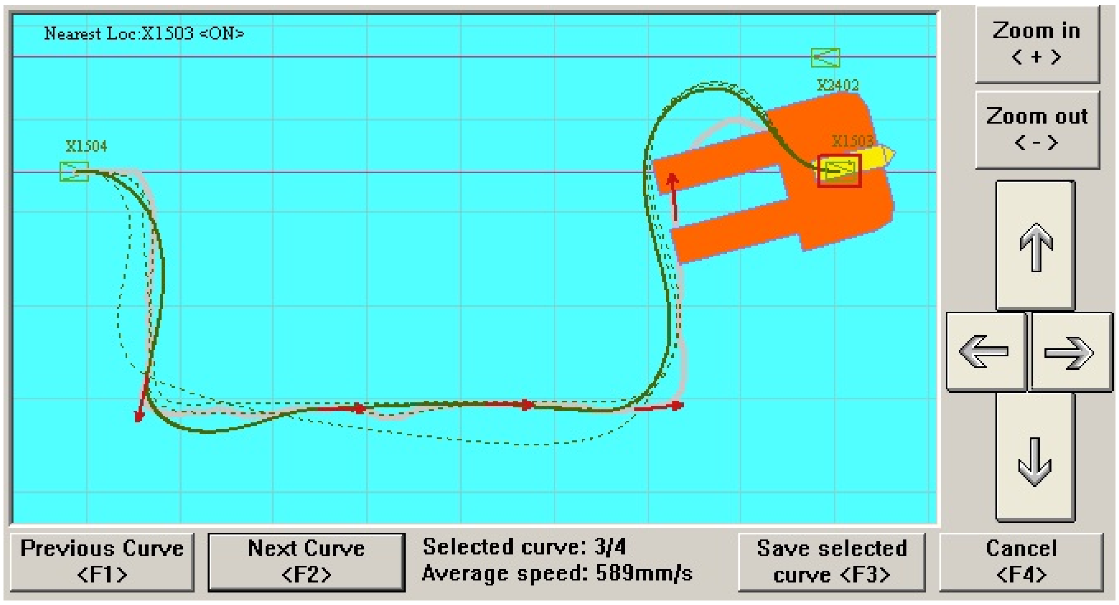
4.2 Program flow

The application is able to handle several different scenarios and allows the user to make various choices depending on the character of the obstacle and the following path. This section describes the general flow of the program as well as the scenarios taken into consideration. Note that the option of going back to the initial path is always available, the user only needs to remove the generated path.

4.2.1 GUI

The GUI is intended to guide the user through the process of generating a new path around an obstacle. Depending on the position of the AGV the possible starting and ending nodes are suggested. The GUI can be used with a touchscreen device and/or a keyboard.

Figure 4.2 shows an example of what the GUI might show an operator generating a new path between nodes X1504 and X1503. The path driven by the operator is drawn as a thick pink line, the suggested paths are drawn in green color with the thicker one being the one chosen at this moment and the dashed ones the other alternatives. At the bottom of the GUI information about the selected path is provided: in this case it is the third path out of four available and the average speed the AGV can uphold on it is 589 $\frac{mm}{s}$. The red arrows along the path show where the operator chose to take a sample (more about the sampling process can be found in section 4.2.2) and the direction of the steering wheel at that point.



Selected curve: 3/4
Average speed: 589mm/s

Figure 4.2: An example of what the GUI might show when operator tries to generate a new path between nodes X1504 and X1503. The thick green curve is the currently selected one, and information at the bottom of the GUI gives some details about it: it is the third curve of four and the average speed the AGV can uphold on it is 589 $\frac{mm}{s}$.

By pushing the `Save selected curve` button the user agrees to use the currently selected path instead of the predefined one (predefined paths are shown in violet), and all the other suggested paths are dismissed. The process can always be canceled, by pressing `Cancel`, in which case the user is returned to the main menu. The arrows on the right, as well as zooming-options, are used to position the map showing the paths, AGV and nodes.

4.2.2 Sampling process

Two kinds of data sampling can occur when the operator manually moves the AGV around the obstacle. One is generated automatically every centimeter ($0.01m$) and achieves a close following of the path, while the other is initiated by the user by pressing the `Take sample` button in the GUI or `Bumper bypass` on the joystick.

When manually moving the AGV the operator might realize that the chosen way is not appropriate. If he/she has sampled data along the erroneous way the button `Remove last sample` allows the last sample to be removed. By pushing the button several times several samples can be removed. This procedure does not effect the automatically sampled data.

After realizing his/hers mistake, and possibly removing some sample points, the operator would probably chose to move backwards, until the vehicle is back on the right track. As long as the reversing is not exaggerated, i.e. mostly follows the way the AGV took before, the automatically sampled data will adapt and the faulty data points will be removed. The process is initiated as the vehicle starts reversing and it removes sample data at the same rate as it previously collected it.

An alternative way of handling the choice of a erroneous path is to start over from the beginning. In this case the application should simply be cancelled and started again when the desired initial position is reached.

4.2.3 Path adaption

The paths generated from the automatically sampled data, as well as the paths generated from the manually sampled points, might initially not be feasible. Therefore, some adaptions are performed before the paths are presented to the user. Figure 4.3 illustrates the steps performed on the sample data in order to ensure feasibility. Algorithm 7, 8, 10, 12, the end segment adaption, as well as the filtering procedure are described in this section.

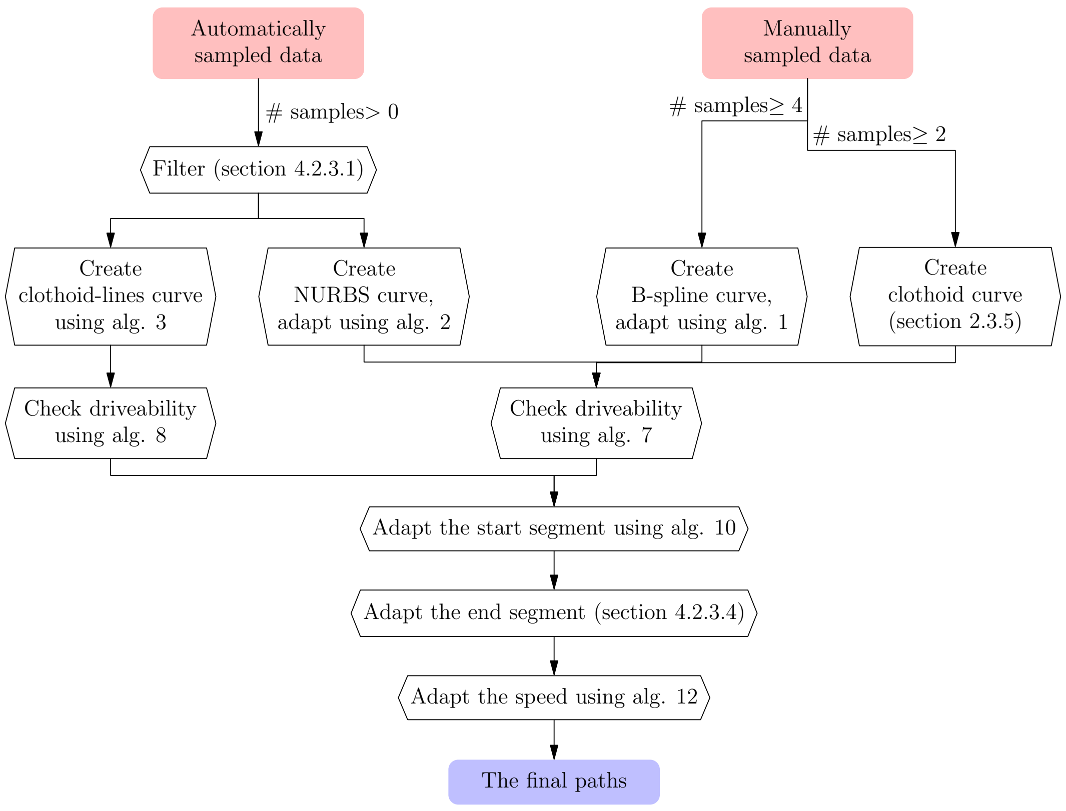

Automatically sampled data
Manually sampled data
# samples> 0
# samples≥ 4
# samples≥ 2
Filter (section 4.2.3.1)
Create clothoid-lines curve using alg. 3
Create NURBS curve, adapt using alg. 2
Create B-spline curve, adapt using alg. 1
Create clothoid curve (section 2.3.5)
Check driveability using alg. 8
Check driveability using alg. 7
Adapt the start segment using alg. 10
Adapt the end segment (section 4.2.3.4)
Adapt the speed using alg. 12
The final paths

Figure 4.3: Flowchart illustrating steps needed to transform the input data (two sets of sample data, red blocks in the figure) to the final paths (blue block in the figure). Depending on the data available up to four paths are generated.

Filtering of the automatically sampled data

Two kinds of issues need to be taken in consideration when preparing the sample data for path generation. First, any loops that might have come to be during the record need to be filtered out. A common reason for loops is when the operator chooses to move backwards during the manual move around the obstacle. Moves like this are considered unintentional and paths should not be generated from the samples containing loops. If the operator wants to make an intentional loop manual samples need to be recorded. The manual samples are not subject to filtering.

The loops are filtered using algorithm 6. The algorithm finds points of intersection along the path and removes all data that lies between those. The segments mentioned in the algorithm are simply lines connecting two successive sample points.

Algorithm 6 Filtering out loops from the automatically sampled data

```
1: while $i <$ number of sample points do
2:     if segment $i$ intersects with any other segment $j$ then
3:         $i \leftarrow j$
4:     end if
5:     Save segment $i$ to the new sample vector
6:     $i \leftarrow i + 1$
7: end while
```

The second issue is that the operator, because of inexperience or for other reasons, might move jerkily when moving the AGV around the obstacle. Using this sample data without any filtering might generate paths that cause excessive actuation and force the AGV to slow down along the segment. This is solved by filtering the data with a normalized Gaussian window.

Figure 4.4 shows an example of the effect the filtering procedure has on the sample data. In this case a loop was formed when the operator chose to move backwards at one occasion. It is deemed very unlikely that the operator wants the generated path to go around in a loop, therefore, the loop is filtered out. Besides, the overall wave-like motion pattern was not intended, but happened because of the operator's inexperience. This too is deemed as a flaw, which the implemented filter smooths out.

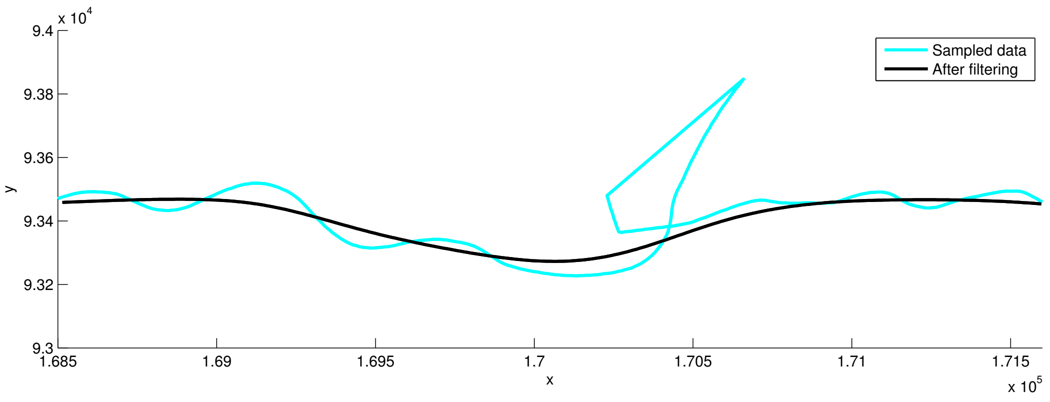


Figure 4.4: The effect the filtering procedure has on the sample data. As the figure shows the probably unintentional loop as well as the wave-like movement along the path are smoothed out.

Curvature adaption along the path

Chapter 3 shows that there exist some limitations on the path. To deal with the mechanical limitations of the steering wheel algorithm 7 is applied to all types of curves except for clothoid-lines curve. The algorithm uses the algorithm 2 to soften all infeasible sharp turns. Figure 4.5 shows an example of the procedure.

Algorithm 7 Dealing with infeasible paths

```
1: while $Path$ not feasible according to algortihm 5 do
2:     $Samples \leftarrow$ equally spaced points every 20cm of $Path$
3:     Apply algorithm 2 to $Samples$ to soften sharp corners
4:     $Path \leftarrow$ the NURBS curve generated using $Samples$
5: end while
```

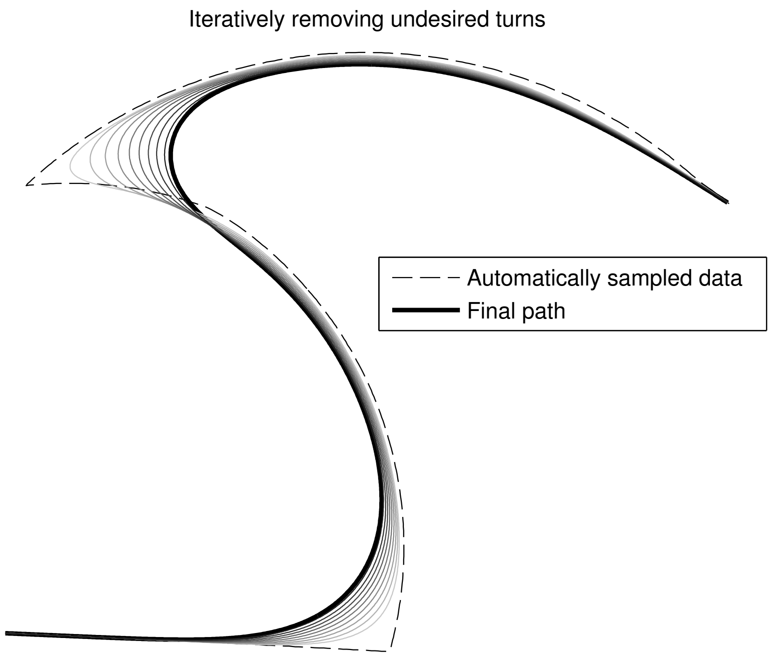


Figure 4.5: Applying algorithm 5 to a path in order to remove infeasible turns. The dashed line is the sample data, used for the first approximation. The following steps of iteration are drawn in gray, the shade of the color representing when they were done. Light color means that the curve comes from a step in the beginning of the iteration process. The final curve, thick black line, is a feasible curve which still follows the sample data well.

As the method involves transforming the data by re-sampling it as a NURBS curve the straight lines in the clothoid-lines curve would be lost. Therefore another method is used to deal with curvature constraints on those curves. It is presented in algorithm 8 and shortens the identified lines until a clothoid with sufficient curvature can be inserted between every pair of lines.

Algorithm 8 Dealing with infeasible curvatures for clothoids-lines curves

```
1: while $Path$ not feasible according to algorithm 5 do
2:     Find out between which lines the sugested path is too sharp
3:     Increase the length of the turn by shortening both lines
4:     if any of the two lines is too short then
5:         Remove it completely
6:     end if
7:     $Path \leftarrow$ assemble again by connecting the new lines using clothoids
8: end while
```

Another issue described in chapter 3 concerns the turning radius causing greater wear on the vehicle and to the floor. From figure 3.2 the undesired steering angle, α_{bad}, can be derived. By applying algorithm 5 it can be found whether α_{bad} is held continuously for at least 10 cm. If segments this long are found a warning is posted that the path includes parts which might cause extra wear on the vehicle and the floor. The operator can then decide whether to keep the path anyway or maybe try driving again. Note that even if one of the paths has a bad segment this does not imply that the other paths have it as well.

Adaption at the beginning of the curve

If the operator starts the application before the start node, all points prior to the start node need to be removed. Algorithm 9 explains the procedure of finding the first local minimum with respect to the distance between the curve and the node. The points are evaluated starting from the first point on the curve, P_1, and moving towards the last, P_{end}. The process stops as soon as the first minimum is reached, which allows the operator to move the AGV close to the start node later on in the sampling process.

Algorithm 9 Deleting curve data prior to the start node

```
1:  $Distance \leftarrow$ distance from $P_1$ to the start node
2:  $LastDistance \leftarrow Distance$
3:  for $i = 2 : end$ do
4:      $Distance \leftarrow$ distance from $P_i$ to the start node
5:      if $Distance < LastDistance$ then
6:          $LastDistance \leftarrow Distance$
7:      else
8:          Remove all points up to $P_{i-1}$
9:          return
10:     end if
11: end for
```

To ensure a driveable path from the start node while keeping as much of the operator's sample data as possible algorithm 10 is used. The algorithm tries the possibility of creating a feasible path from the start node to a point along the curve, using a clothoid segment. The point on the curve is initially located four wheelbases away from the start node. The distance is gradually decreased until the start node is reached. The point located closest to the node, while ensuring a feasible path, is chosen as the starting point on the curve. The output from the algorithm is $FinalPath$, which contains the original path, but with the first segment replaced by a clothoid. Note that there is no guarantee that a feasible path can be created, however the initial radius of four wheelbases should be enough for all but the most extreme cases. In the rare case that the variable $FinalPath$ is returned empty no path will be generated.

Algorithm 10 Ensuring a drivable path from the start node

```
1: Remove points before the start node according to algoritm 9
2: $Radius \leftarrow 4.0$
3: $FinalPath \leftarrow [\quad]$
4: while $Radius > 0$ do
5:     Remove points within $Radius * WheelBase$ from start node
6:     if points left on curve then
7:         Connect a clothoid from the start node to the curve
8:         if the combined path is driveable according to algorithm 5 then
9:             $FinalPath \leftarrow$ the combined path
10:        end if
11:    end if
12:    $Radius \leftarrow Radius - 0.2$
13: end while
14: return $FinalPath$
```

Algorithm 10 is applicable to all types of curves and guarantees a smooth transition from the previous part of the path to the newly generated one. It also ensures that the tangent direction of the first point of the path matches the heading of the start node, guaranteeing G^1 continuity at the joint.

Adaption at the end

At the very beginning of the path generation, before the data is sampled, the operator has the opportunity to choose how accurately the heading of the AGV at the end of the newly generated path needs to match the end-node's predefined heading. High accuracy is important if the following path, for instance, has high speed, or includes a docking station.

In order to ensure that the final heading angle requested by the operator is achieved, the area around the end-node is encircled by two circles. The inner circle has a fixed radius of one wheelbase while the outer circle's radius is initially four wheelbases.

The outer circle is treated similarly to algorithm 10, the differences being that the *Radius* is allowed to shrink to one wheelbase instead of all the way to the end-node and that the points are removed from the end of the curve instead of from the beginning. Note that this also ensures that the operator passing the end-node is not an issue, as any points located after the node will be removed.

When the sample data inside the outer circle is removed, as stated in algorithm 10, a new point is inserted at the perimeter of the inner circle. The new point is initially located along the line between the end-node and the currently last point on the curve, with a heading pointing towards the end-node.

This means that there now exists two segments to generate clothoids between, one from the outer circle to the inner circle, and one from the inner circle to the end-node. A correct end heading is ensured by iteratively moving the point on the inner circle to different locations according to algorithm 11. The procedure is repeated for every new $Radius$ of the outer circle. Figure 4.6 shows an example of the approach of adapting the path at the end-node.

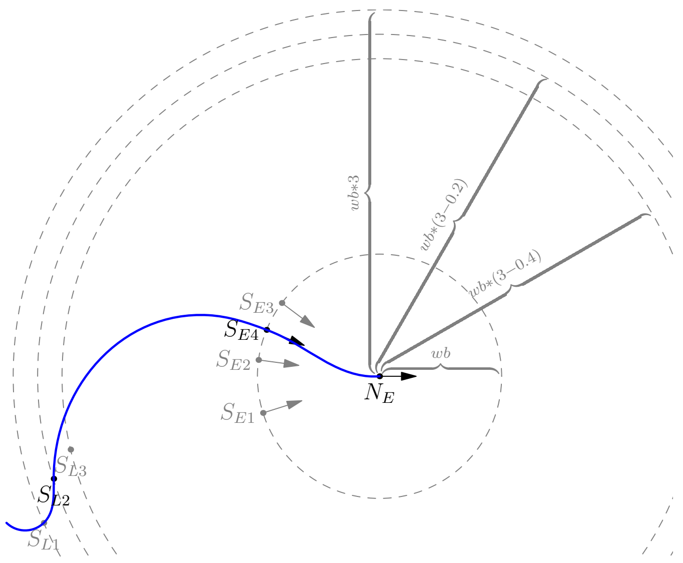


Figure 4.6: Procedure for securing a final heading angle required by the operator. The figure shows the final curve after the outer iteration has occurred three times (point S_{Li} is the last point on the path for the respective iteration. Only the last inner iteration is shown, with points S_{Ei} presenting the movement of the extra sample point, placed on the perimeter of a smaller circle. As the figure shows the direction of the movement was changed after three iterations and the step size halved. N_E (the end-node), S_{L2} and S_{E4} are drawn black in order to emphasize that they are the points shaping the final path.

Algorithm 11 presents the iteration procedure for the smaller (inner) circle around the end-node. $MaxIteration$ is the number of iterations that are allowed to take place at the most before the procedure fails; $Path$ is the curve from the start node to the outer circle; $Candidate$ is the combination of the path and the suggested ending, used to check whether the desired end-heading of the AGV is achieved; wb is as usually the wheelbase; $N_E = (x_E,y_E,\theta_E)$ and $S_L = (x_L,y_L,\theta_L)$ are the coordinates of the end-node respectively the currently last point of the curve, including tangent direction; $Tolerance$ is defined by the operator and is the maximum allowed difference between the heading of the AGV at the end-node and the predefined heading at that point.

Algorithm 11 Iteration of the inner circle around the end-node

```
1: $Angle \leftarrow$ angle of the vector pointing from $S_L$ to $N_E$
2: $StepSize \leftarrow 15^o$
3: $Iteration \leftarrow 1$
4: while $Iteration < MaxIteration$ do
5:     $x \leftarrow x_E - cos(Angle) * wb$
6:     $y \leftarrow y_E - sin(Angle) * wb$
7:     $S_E \leftarrow$ (x, y, Angle)
8:     $Part1 \leftarrow$ a clothoid from $S_L$ to $S_E$
9:     $Part2 \leftarrow$ a clothoid from $S_E$ to $N_E$
10:    $Candidate \leftarrow [Path, Part1, Part2]$
11:    Check end heading of $Candidate$ using algorithm 5
12:    if $|\theta_E$-end heading$| < Tolerance$ then
13:        return Candidate
14:    else
15:        $\mu \leftarrow sign(endheading - \theta_E)$
16:        if $\mu$ changed compared to last iteration then
17:            $StepSize \leftarrow StepSize/2$
18:        end if
19:        $Angle \leftarrow Angle + \mu * StepSize$
20:    end if
21:    $Iteration \leftarrow Iteration + 1$
22: end while
```

4.2.4 Speed adjustments

There are, as mentioned in chapter 3, several issues to consider when deciding the speed the AGV should keep along the path. The speed is also essential when evaluating whether the path is feasible or not, as algorithm 5 shows.

To ensure the feasibility of the path and a good match to the sampled path it is therefore appropriate to start with a low speed. When a drivable path has been generated the maximum allowed speed at every point on the path can be calculated using equations 3.3 and 3.4. The speed at every point is set to the lower of the two.

As there is no use trying to change the speed of AGV at every point, the points are combined into longer segments using algorithm 12. The algorithm ensures that the segments are long enough to keep the current speed for $MinTime$ seconds. $MaxSpeed$ contains the maximum speed allowed for each point on the path. The segment speed is always chosen as the minimum of the current segment's speed ($FinalSpeed_j$) and $MaxSpeed(i)$. *acceleration* and *deceleration* are AGV specific constants describing the motor acceleration and retardation in $\frac{mm}{s^2}$. *MinSpeedDiff* is a user set variable for tuning how small speed differences should be ignored.

Algorithm 12 Combining individual speeds into speed segments

```
1: $j \leftarrow 1$
2: $FinalSpeed_j \leftarrow MaxSpeed(1)$
3: $PrevSpeed \leftarrow FinalSpeed_j$
4: $Distance \leftarrow 0$
5: for $i \leftarrow 2$ up to number of points on the path do
6:     $v_1 \leftarrow FinalSpeed_j$
7:     $v_2 \leftarrow MaxSpeed(i)$
8:     if $v_1 > PrevSpeed$ then
9:         $dist_{acc} \leftarrow (v_1 - PrevSpeed)/acceleration$
10:    else
11:        $dist_{acc} \leftarrow 0$
12:    end if
13:    if $v_1 > v_2$ then
14:        $dist_{dec} \leftarrow (v_1 - v_2)/deceleration$
15:    else
16:        $dist_{dec} \leftarrow 0$
17:    end if
18:    $Distance \leftarrow Distance + DistanceFromPreviousPoint(i)$
19:    $MinDistance \leftarrow MinTime \times v_1$
20:    $DistanceLeft \leftarrow Distance - dist_{acc} - dist_{dec}$
21:    $SpeedChange \leftarrow abs(v_2 - v_1)$
22:    if $DistanceLeft > MinDistance$ && $SpeedChange > MinSpeedDiff$ then
23:        $j \leftarrow j + 1$
24:        $FinalSpeed_j \leftarrow v_2$
25:        $Distance \leftarrow 0$
26:    else
27:        $FinalSpeed_j \leftarrow min(v_1, v_2)$
28:    end if
29: end for
```

When presenting the paths to the operator the average speed of each path is shown. This way the operator has more freedom of choosing a path convenient for the specific scenario he/she is trying to resolve.

5

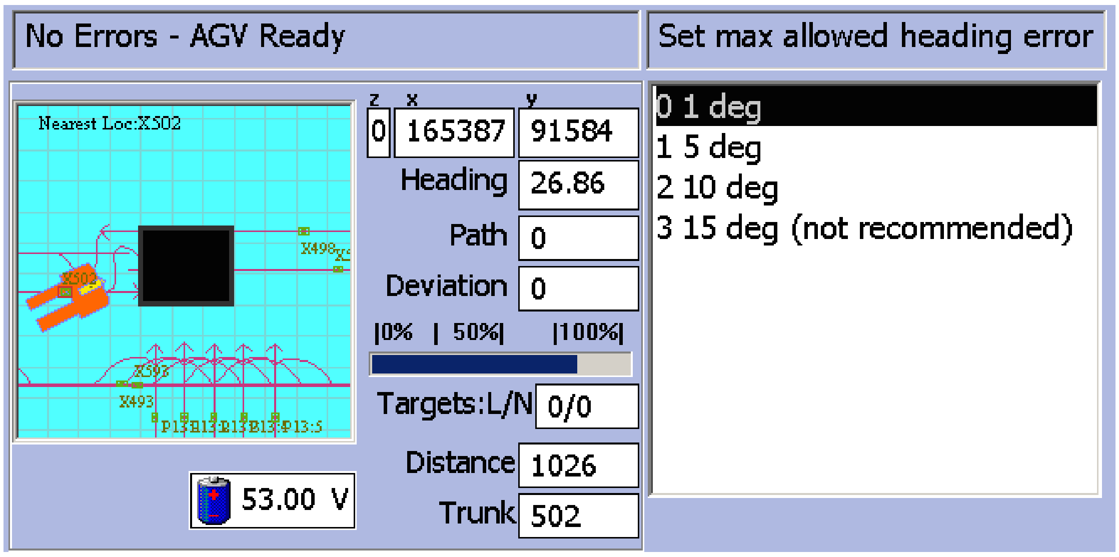
Results

The results are presented in the form of a possible situation that could occur at an industry where it would be beneficial to apply this application. The scenario is explained in text, and figures are used to illustrate the results.

5.1 Test scenario

The following scenario is applied: an operator finds that an AGV has come to a stop in front of an obstacle between nodes X502 and X501. Operator realizes that removing the obstacle is problematic and chooses to create a new path for the AGV. She moves the AGV back to node X502 and from there launches the application.

The path after X501 is a straight line, meaning that the segment has a high predefined speed. Because of this it is important that the AGV approaches the node with a correct heading, or the deviation from the path might become too large for the controller to handle. The operator therefore chooses the allowed heading error to be less than 1 degree. Figure 5.1 presents how the situation is shown in the interface of the AGV. Note that the black box in the figure is placed there for the readers convenience, in the real application the obstacle is not shown as the application has no way of knowing where the obstacle is.


No Errors - AGV Ready

Set max allowed heading error

z 0 x 165387 y 91584

Heading 26.86

Path 0

Deviation 0

|0% | 50%| |100%|

Targets:L/N 0/0

Distance 1026

Trunk 502

53.00 V

0 1 deg
1 5 deg
2 10 deg
3 15 deg (not recommended)

Figure 5.1: GUI in the beginning of the test scenario. The operator has found an obstacle in the way of the AGV (the black box in the figure) and decided to create a new path for the AGV. She therefore placed AGV at the start node and is now making the final choices before moving the AGV around the obstacle.

5.2 Moving around the obstacle

Now the operator has to move around the obstacle. She chooses to not only rely on the automatically sampled data but takes some samples of her own as well. As she has not used the joystick for some time the path she takes is a bit jerky and disjointed (at some points she realizes that she moves to close to the obstacle and has to move backwards). Never the less she manages to drive all the way to the end-node, helped by the fact that it is marked red in the layout, as she during the move around the obstacle forgot exactly where she was heading. Figure 5.2 is a snapshot of the GUI at that moment.

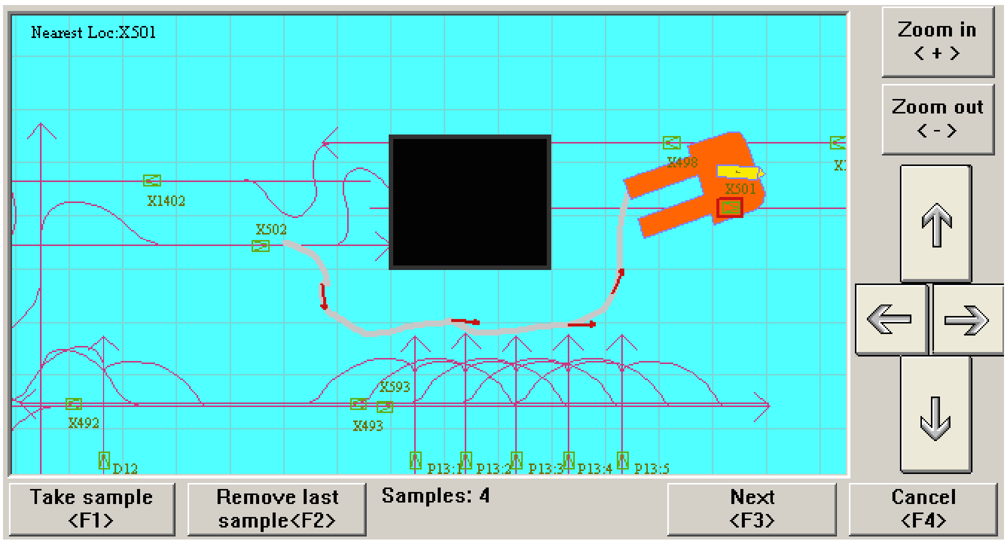


Figure 5.2: GUI after the operator has driven the vehicle around the obstacle. The pink thick line is the path of the AGV, the red arrows are where the operator chose to take samples. The end node is marked with red. Once again, the black box symbolizing the obstacle is placed there for the reader's convenience and is not visible in real application.

5.3 Calculating the paths

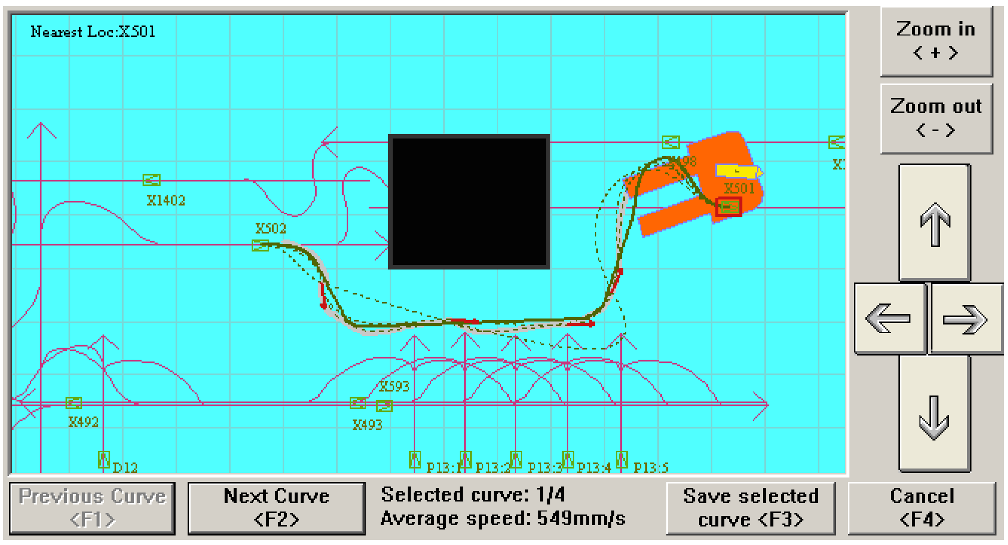
The operator now presses the `Next` button whereupon, in a few seconds, the GUI looks as figure 5.3 shows. Four paths have been calculated, with corresponding speeds. All presented paths are guaranteed to be feasible and safe to drive.

The first path, shown in figure 5.3, is the one built up of clothoids and lines and uses the automatically sampled data as input (see chapter 2 for more details about the different types of paths). The path ensures long straight lines, which makes the movement of the AGV along the path smooth. The speed along the straight lines is high, but to make the usually quite sharp turns between two lines (the connection is made by clothoids) a rather slow speed is needed. After the paths are generated this path is always the one initially selected.

The second path is too build from the automatically sampled data, but uses NURBS to create the curve. This curve follows the sample data closely, which means that the path might be a bit jerky, even though the data is filtered.

The third and fourth paths use user-sampled data and are built from B-splines and clothoids, respectively. Those paths are smoother than the one build with NURBS as fewer samples are used. The main characteristic of the B-splines path is that is tries to keep the entire path outside the sample points.

The clothoid path ensures that all the user-sampled points are interpolated and that the direction of movement, illustrated by the red arrows in figure 5.3, is kept. This means that the vehicle moves to and away from a sample in the same direction as it did when the operator drove. This feature of the clothoid curve can be used to create paths with complicated shapes, as long as the operator remembers to sample often enough.



Selected curve: 1/4
Average speed: 549mm/s

Figure 5.3: The GUI after the paths have been calculated. By pressing `Save selected curve` the path chosen at that moment is saved as the new path of the AGV. When choosing the right path for the specific situation it might be appropriate to not only consider the shape of the path but also the average speed (displayed in the GUI) and whether a risk for extra wear of the vehicle and floor exists. If the risk exists it will be shown as a warning under the information about the average speed. Note that all the suggested paths look almost identical at the start and the end. This is due to the fact that the algorithms for start- and end-node adaption are the same for all curves.

All the generated paths are guaranteed to start at the start-node and end at the end-node. Because of the enforced end heading and the same start heading all the curves look very similar at the start and end.

5.4 Updating the layout

In this scenario the operator decides that the first path, the one consisting of clothoids and lines, is favorable. When operator has made the decision she presses `Save selected curve` and the new path is saved in the layout. Figure 5.4 shows the layout as it will look from now on, until a new path between the same two nodes is generated or the operator decides to go back to the predefined path by removing the new one. All the other suggested paths are dismissed.

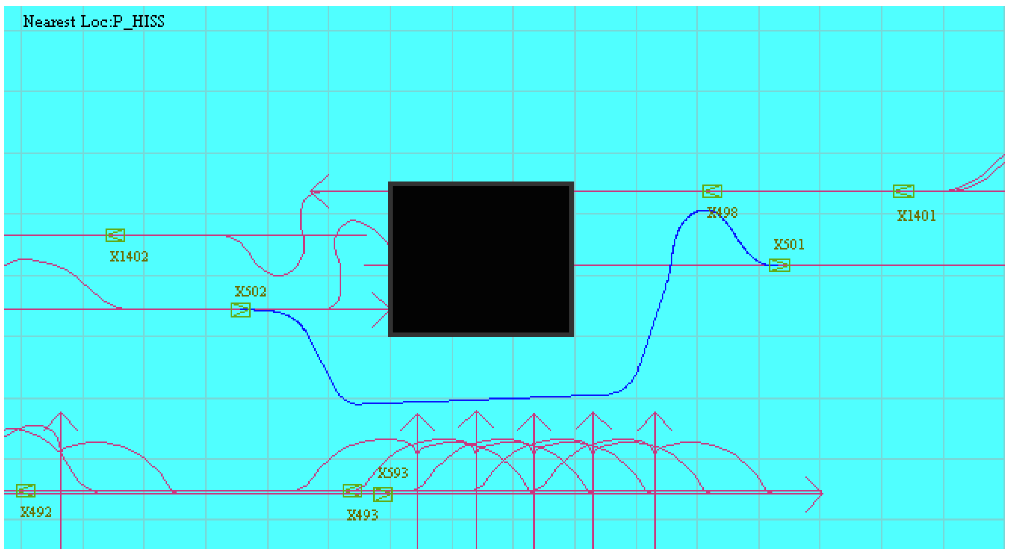


Figure 5.4: The figure shows what the layout looks like after a path has been chosen. This path will be the one AGV drives until a new path between the same nodes is generated or the operator chooses to go back to the predefined path by removing the new one.

Next time the AGV reaches the node X502 operator chooses to walk along it to make sure that the obstacle is cleared and if not to be able to interfere. When the AGV manages to complete the path without any problems the operator goes back to the tasks she performed before she found the obstacle in AGV's path. The end of test scenario.

5.5 The speed along the path

Figure 5.5 presents the speed segments corresponding to the chosen path, compared to the speed recorded when driving it. The actual speed differs a bit as it includes ramps, used to make sure that the AGV decelerates in time if the speed of the following segment is lower than the one held at the current segment. The suggested speed segments are calculated to ensure a safe drive, i.e. there is no tilt risk and the path will not turn faster than the AGV is capable of keeping up with.

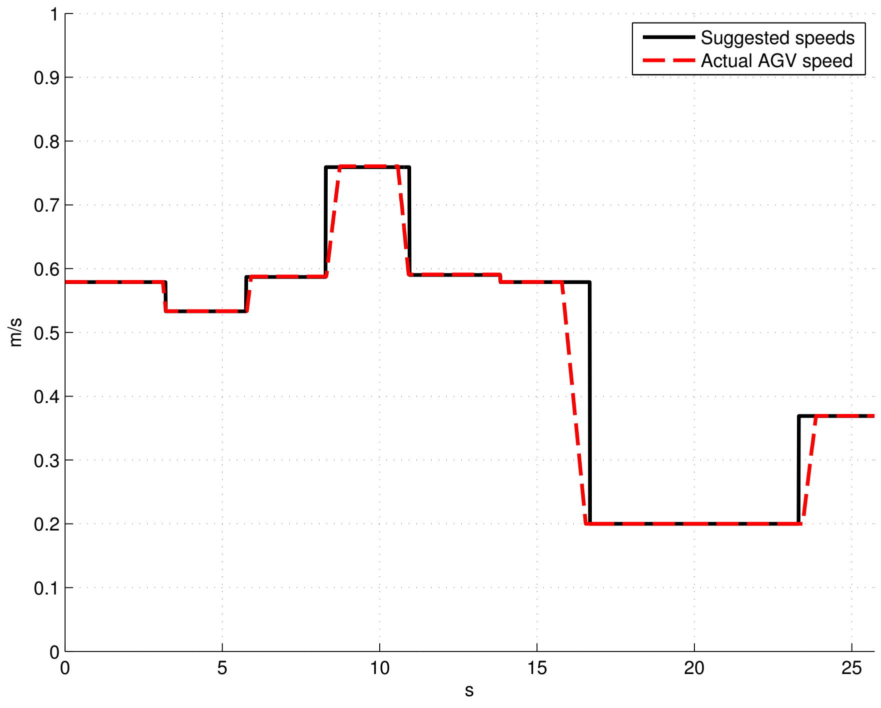


Figure 5.5: A graph for comparison of the by application suggested speed segments and the speed actually held by the AGV. The actual speed includes ramps to ensure that the AGV does not approach a slow segment with too high speed.

5.6 Comparison of execution times of different paths

The generation of the different paths does not happen instantaneously, partly due to the low priority of the execution thread and partly due to the computational difficulty of the different curves. Table 5.1 presents an overview of the execution times of the different processes in the application. The length of the short path is 5.56 m and 4 user-samples were collected. The length of the long path is approximately 13.7 m and 8 user-samples were collected. The data was collected during five tries following exactly the same path (of respective length), but the placement of the user-samples differed a bit, as they were placed manually. The process of filtering data includes both filters mentioned in section 4.2.3.

Process	Execution time [ms]	
	Short path	*Long path*
Filtering of data	352 (29.7)	1 617 (74.8)
Clothoids-lines	2 905 (35.4)	4 954 (211.1)
NURBS	1 851 (32.6)	4 302 (406.7)
Clothoids	1 992 (98.9)	6 499 (594.8)
B-spline	3 207 (276.7)	5 344 (667.8)
Total time [ms]	10 307	22 716

Table 5.1: The average execution times for processes in the application. Values inside the parentheses give the standard deviation between the different tries.

Standard deviation, σ, is included as a measurement of stability of the execution times. The deviation is calculated using equation (5.1), where n stands for the number of data points, x_i is the data point i and $\bar{x}$ is the average execution time, presented in table 5.1.

$$\sigma = \left(\frac{1}{n-1} \sum_{i-1}^{n} (x_i - \bar{x})^2 \right)^{1/2} \tag{5.1}$$

The total time for the paths calculations are approximately 10 respectively 23 seconds. Even when driving a longer path (about 50 m) the paths are calculated within one minute.

6

Evaluation of the implementation and the results

During this project we managed to create an application for generating new paths between two consecutive nodes. Any operator familiar with AGVE's vehicles should be able to create a new path following instructions on the GUI. The speed along the paths is calculated to be as fast as possible, while still guaranteeing a safe drive, and the application is well integrated with the existing software. We therefore consider the objective of this project reached.

The chapter starts by putting our application in a larger context, by discussing the sustainability issues affected by it. The rest of the chapter follows the general structure of the report, discussing and evaluating different design choices made. Of course many other options for solving the same problems exist and it is possible that those are superior to the ones suggested by us. Finding and implementing them is, however, deemed outside the scope of this project.

6.1 Sustainable development

Sustainability includes economical, social and environmental aspects. This application, when implemented, affects all of these.

The application should result in a larger number of AGVs in industries, exchanging the manual forklifts. As manual forklifts are today one of the greatest causes of injury at industry (see chapter 1) this affects the social aspect, as well as economical. The economical effect comes from the well known fact that injuries are expensive for the society.

Other economical benefits come from the ability to quickly reconfigure the vehicles, which allows workers to swiftly react to changes in the working area thus reducing the stand-still time in the production. This also results in environmental benefits, as engineers don't need to travel to the site when a reconfiguration is needed. This is especially important for a company such as AGVE, which has customers all over the world.

The more flexible layout definition allows any worker on site to adapt the paths, not only if something blocks the path but also if a worker sees a way of improving the path. This way small improvements can be made continuously, resulting in a better overall production flow.

6.2 Bézier curves

As mentioned in section 2.4 the Bézier curves are hard to adapt to the given sample points. Many different cases are possible, every one of them had to be tweaked by hand, making it impossible to guarantee that all were actually covered.

Another point of concern was the time the computation of the different curves took. Even though Bézier curve is not the computationally hardest one it still takes time to complete. Besides, four curves were deemed to give enough options to allow the operator to get what he/she wanted.

6.3 NURBS and B-splines

A person familiar with NURBS from before, or one who has read this report carefully, might notice that NURBS used in the implementation are actually NUBS. R, which stands for "rational", would mean including different weighting for some of the control points, but it was concluded that adjustment of weights was not a good enough way of adapting curves for our purpose. Weights are used when few sampled points are available and the curve is to follow a predefined shape. As we had many points and no way of knowing exactly what shape the user intended it was deemed better to adapt control points (see algorithm 2) and leave rationalization out of it.

It might also be argued that B-splines used in this project were very similar to the NU(R)BS and therefore maybe unnecessary. The only difference in calculations is in the shaping of the knot vector, which is nonuniform for NURBS and uniform for B-splines. B-splines were kept because of how we used them. In this project B-splines are used to evaluate user-sampled data, while NURBS initially only evaluate the automatically sampled data. This means that the adaption relevant for respectively curve type is different. The generated NURBS only attempt to follow the curve as closely as it can feasibly be done and skip the sharp corners, while the generated B-splines try to make sure that as much of the final curve as possible is kept outside the obstacle area (obstacle area is area inside the user-sampled data).

6.4 Clothoids

Bertolazzi and Frego write in their draft to the article ”Fast and accurate G^1 fitting of clothoid curves” [22] that ”it is well known that the best or most pleasing curve is the clothoid”. The formulation in the final version is less powerful (”it is well known that clothoids are extremely useful”) but nevertheless it is apparent that they consider clothoid to be a very important curve. During this project we came to agree with them.

A great advantage of clothoids is how readily they merge with other curves. By inserting the tangent direction and the coordinates of the joint into the clothoid-calculations a clothoid ensuring G^1 continuity emerges. This fact can also be used, which we did in section 2.4.3, when designing ”fast” paths. As the fastest speed can be achieved if the curve is perfectly straight we let long lines initially approximate the curve, and use clothoids to join them. The resulting curve is a good approximation of the automatically sampled data *and* usually the fastest one of the suggested curves.

The only disadvantage clothoids have is that they are quite computationally heavy. By using algorithm discussed in section 2.3.5 and making sure that the number of inserted points is not larger than necessary this disadvantage disappeared. The resulting clothoids are fast, easy to use and, as our experience shows, without any flaws. During this project we several times tried to blame clothoids for problems we had with path generation, but every time it turned out that the fault lay elsewhere.

6.5 Algorithm 3

Inspiration to let algorithm 3 find as long lines as possible, without enforcing a predefined length on them, came from the article by Liu et al. where Moving Least Square method was mentioned[28]. As their method did not ensure exact interpolation for the first and last sample points in the attempted line, different ways of conducting linearity measures were studied ([29] presents several different methods adapted for ordered point sets). The method decided on is somewhat akin to the Triangle Sides Ratio method discussed by Stojmenovic and Nayak[29], but instead of only checking one point along the curve the distance between every point on the line and the corresponding point on the curve is checked. In retrospect, the requirement on the first and last point on the attempted line could probably be dropped to produce longer lines.

It was decided not to force any circles into the path as those need to be connected to the lines using clothoids, which would require two clothoids instead of one to connect two lines together (with a circle in between). Besides, tries were made with paths that included circles versus paths that did not include circles, and no pronounced difference was found, expect for increased computational difficulty.

Algorithm 3 has to check all following points for every point on the curve, making its complexity $O(n^2)$. This makes it very computationally demanding, especially if the generated path is long. Table 5.1 shows a comparison between the execution times for different processes in the application. For the longer path the execution time of the clothoids-lines curve has increased considerably, making it by far the slowest process in the application. The benefits of this path are though considered worth the extra time.

6.6 Sampling

In section 4.2.2 it is mentioned that the sampling frequency is one sample every 10 mm. This choice is motivated by the fact that the control loop in the main software in the simulator, the one that makes sure that the AGV keeps its path, is initiated every 10 ms. As the maximum speed of the AGV is approximately 1000 $\frac{mm}{s}$ a sample every 10 mm is as fast as we can sample.

It was later discovered that the control loop in the real-life application is initiated every 2 ms, implying that a faster sampling frequency is possible. The initially suggested sample rate did though work very well during the test drives and it was decided to not make any further adjustments in this area.

6.7 Filtering

As mentioned in section 4.2.3 the automatically sampled data might need filtering. We implemented two different filters. First the loops were deleted using algorithm 6. The presented algorithm is not optimal in the sense of computational efficiency, but is easily programmed and, as table 5.1 illustrates, fast enough to be used in this project. It might be appropriate to use the Sweep Line algorithm [30] instead, but to implement it is deemed outside the scope of this project.

The second filter is a Gaussian window, applied to remove smaller irregularities in the data. It is possible that other filters would have worked better, but as the Gaussian window was easy to implement and gave satisfactory results we chose to not study this any further.

6.8 Algorithm 5

An important part of this project is algorithm 5. It is used to test the driveability of the paths and no results would have been achieved without it. It is, however, an approximation and, even though it worked very well during this project, it might be appropriate to refine it. For instance keeping the time dependent step and setting it to the cycle time of the control loop, instead of making the change described in section 3.2, might give a more accurate simulation.

A shorter timestep would increase the precision of the simulation, which would especially affect the maximum velocity. Since the velocity calculation depends on a derivative, small changes in the precision lead to large errors. However, the procedure for speed calculations implemented today ensures that the lowest speed over a segment is used, which means that even if there are inaccuracies the final speed is not likely to be unsafe.

It might also be possible to replace the entire simulation procedure by appropriate mathematical relations, formulating the path generation as an optimization problem instead. This is discussed in more detail in section 7.1.

6.9 Adaption at the end

As figure 4.3 shows the end-node clothoids are appended to the path after the rest of it has been generated. This is possible as the clothoid calculation makes sure that the transition between the previous curve and the clothoid segment is G^1 continuous, thus enabling us to adapt the end without having to recalculate the entire curve for every iteration.

The iterative procedure for ensuring a tight and smooth turn to the end-node described in algorithm 11 takes about 6-10 iterations to complete. The effectiveness of it makes it possible to try out many different scenarios in a very short time span, which we consider a great success.

By setting a very low speed (200 $\frac{mm}{s}$) for the last segment it is possible to generate tight turns, thus ensuring that as much as possible of the operator generated path is preserved while the transition to the following path is not jeopardized. The design of this procedure is based on the assumption that the preservation of the data is of greater importance than the speed along the path. However, when designing long paths it might be preferable to get as fast, instead of as tight, curves as possible. Our application can easily be adapted to suit that need, but as the arena used for testing was quite small we chose to keep the slow speed and tight turns.

6.10 Execution times

Table 5.1 shows the execution times for different processes in the application. All the processes can be significantly optimized, decreasing the total time needed for paths calculation. The application is, however, not intended to be used very often, making a delay of 20 seconds for a rather long path seem acceptable.

The standard deviation lies in the interval $1 - 12.5\%$, compared to its respective execution time. As the application is not time critical we consider those values acceptable.

The highest percentages are found for B-splines and clothoids, which have iterative solutions, dependent of the placement of the user-samples. As the samples were placed manually it is natural that the standard deviation is larger compared to the automatically sampled data.

Other factors that have an effect on the standard deviation are the scheduling (our application has the lowest priority of all processes in the AGV), the deviation from the start- and end-node which might occur when positioning the AGV and the fact that the positioning system is not 100% accurate.

7

Future work

The ambition of this project was to present a reliable final application, easy to use for any operator experienced with AGVE's vehicles. We regard the ambition reached, but as the aim was to generate an all-around solution some parts of the application are not as well designed as we might want them to be. In this chapter suggestions for future work are presented and explained. Our opinion is that implementing these improvements would enhance the already existing application.

Other types of adjustments to the application that could be made concern the choice of nodes to generate paths to (now limited to the ones directly following the current node) and further integration of this application in the existing systems, for instance by including traffic controller and several vehicles. These adjustments are deemed outside the scope of this project and are therefore not discussed in this chapter.

7.1 Infeasible curvatures

Section 3.2.1 discusses the curvature limitations enforced by the mechanical constraints of the steering actuator. To ensure that the suggested paths are not subjected to these limitations, a simulation procedure, described in algorithm 5, is applied. The solution is neither optimal in the sense of computational efficiency nor perfectly accurate. Instead the problem could be solved by formulating a mathematical relation for finding undesirable turns in a deterministic way. The fact that $\dot{\alpha}$ in algorithm 5 multiplied by a factor $\frac{1}{v}$, is the same as the curvature can be used to formulate an efficient constraint in an optimization problem.

When considering paths generated using clothoids it might be appropriate to solve this using the measure of sharpness[1]. An algorithm for clothoid calculation which takes consideration to sharpness is described in [31]. As the feasible curvatures using their method are calculated with respect to sharpness limitations, instead of shortest possible

[1]Sharpness is the derivative of curvature with respect to distance traveled on the curve

length, the resulting clothoid-segments are longer than the ones used today. This means that the clothoids are less tight, which can result in more easily traveled paths.

7.2 Moving backwards

During the sampling process the operator is allowed to move backwards (this is mentioned in section 4.2.2). While AGV is moving in reverse the automatically sampled data is cleared at the same rate as it was previously sampled. If operator only reverses for a short while and mostly follows the path taken before (but now considered faulty) the clearing is justified. But if the operator chooses to move backwards in some other direction and/or keeps reversing for a long time, the previously sampled data that the operator would *not* like removed might, nevertheless, disappear.

Therefore we suggest that a more effective and fair process of removing data points is designed. It might be proper to let the operator decide how far the data should be cleared, or maybe no clearing should be done at all and instead a smarter algorithm for adapting sample data to a path would solve the problem. Either way, this is a topic for improvement.

7.3 Increased wear on vehicle and floor

The origin of the angle causing increased wear of the vehicle and floor is explained in section 3.2.1. In this project we chose not to focus on eliminating it from the paths, but only warn if a path includes a segment with that turning radius (see section 4.2.3). Our choice is motivated by the facts that segments like this are not a very common occurrence and that a path including these is still drivable. It is though proper to include an action for getting rid of this kind of segments in future implementations.

Bibliography

[1] I. F. Vis, Survey of research in the design and control of automated guided vehicle systems, European Journal of Operational Research 170 (3) (2006) 677 – 709.
URL `http://dx.doi.org/10.1016/j.ejor.2004.09.020`

[2] H. Martínez-Barberá, D. Herrero-Pérez, Autonomous navigation of an automated guided vehicle in industrial environments, Robot. Comput.-Integr. Manuf. 26 (4) (2010) 296–311.
URL `http://dx.doi.org/10.1016/j.rcim.2009.10.003`

[3] Arbetsmiljöverket/Enheten för statistik och analys, Korta arbetsskadefakta (2013, accessed April 26, 2014).
URL `http://www.av.se/statistik/faktarapporter/arbetsskadefakta/2013_04_Truckar.aspx`

[4] D. H. Shin, S. Singh, Path Generation for Robot Vehicles Using Composite Clothoid Segments, Tech. Rep. CMU-RI-TR-90-31, Robotics Institute, Pittsburgh, PA (December 1990).
URL `http://www.dtic.mil/dtic/tr/fulltext/u2/a232947.pdf`

[5] L. Piegl, W. Tiller, The NURBS Book, Monographs in Visual Communication, U.S. Government Printing Office, 1997.
URL `http://books.google.se/books?id=7dqY5dyAwWkC`

[6] B. A. Barsky, A. D. DeRose, Three Characterizations of Geometric Continuity for Parametric Curves, Tech. Rep. UCB/CSD-88-417, EECS Department, University of California, Berkeley (May 1988).
URL `http://www.eecs.berkeley.edu/Pubs/TechRpts/1988/5276.html`

[7] T. Lyche, K. Mørken., Spline methods, draft.
URL `http://heim.ifi.uio.no/~knutm/komp04.pdf#sthash.LRuj8I2S.dpuf`

[8] M. K. Agoston, Computer graphics and geometric modeling : mathematics, Springer, London, 2005.

[9] D. Meek, D. Walton, Clothoid spline transition spirals, Mathematics of Computation 59 (199) (1992) 117–133.

[10] J. Daintith, E. Wright, A dictionary of computing (2008).
URL http://dx.doi.org/10.1093/acref/9780199234004.001.0001

[11] M. K. Agoston, Computer graphics and geometric modeling : implementation and algorithms, Springer, London, 2005.

[12] M. E. Hohmeyer, B. A. Barsky, Rational Continuity: Parametric, Geometric, and Frenet Frame Continuity of Rational Curves, ACM Trans. Graph. 8 (4) (1989) 335–359.
URL http://doi.acm.org.proxy.lib.chalmers.se/10.1145/77269.77274

[13] G. Farin, J. Hoschek, M.-S. Kim, Handbook of Computer Aided Geometric Design, North Holland, 2002.
URL http://www.sciencedirect.com/science/book/9780444511041

[14] C. Cook, C. Ho, The application of spline functions to trajectory generation for computer-controlled manipulators, in: I. Aleksander (Ed.), Computing Techniques for Robots, Springer US, 1985, pp. 101–110.
URL http://dx.doi.org/10.1007/978-1-4684-6861-8_6

[15] K. Petrinec, Z. Kovacic, The Application of Spline Functions and Bézier Curves to AGV Path Planning, in: Industrial Electronics, 2005. ISIE 2005. Proceedings of the IEEE International Symposium on, Vol. 4, 2005, pp. 1453–1458.
URL http://dx.doi.org/10.1109/ISIE.2005.1529146

[16] J. McCrae, K. Singh, Sketching piecewise clothoid curves, Computers & Graphics 33 (4) (2009) 452 – 461.
URL http://dx.doi.org/10.1016/j.cag.2009.05.006

[17] L. Wang, K. Miura, E. Nakamae, T. Yamamoto, T. Wang, An approximation approach of the clothoid curve defined in the interval $[0, \frac{\pi}{2}]$ and its offset by free-form curves, Computer-Aided Design 33 (14) (2001) 1049 – 1058.
URL http://dx.doi.org/10.1016/S0010-4485(00)00142-1

[18] H. P. Moreton, Minimum curvature variation curves, networks, and surfaces for fair free-form shape design, Ph.D. thesis, EECS Department, University of California, Berkeley (Mar 1993).
URL http://www.eecs.berkeley.edu/Pubs/TechRpts/1993/5219.html

[19] L. Piegl, On nurbs: a survey, Computer Graphics and Applications, IEEE 11 (1) (1991) 55–71.
URL http://dx.doi.org/10.1109/38.67702

[20] J. Wallace, Nurbs in a nutshell, Manufacturing Engineering 119 (4) (1997) 92–98.
URL http://search.proquest.com/docview/219699539?accountid=10041

[21] M.Dahari, Y. Liu, A Review of Auto-Guided-Vehicles Routing Algorithms, Advanced Materials Research 479 - 481 (2012) 443–456.
URL `http://www.scientific.net/AMR.479-481.443`

[22] E. Bertolazzi, M. Frego, G^1 fitting with clothoids, Mathematical Methods in the Applied Sciences (2014) n/a–n/a.
URL `http://dx.doi.org/10.1002/mma.3114`

[23] I. Baran, J. Lehtinen, J. Popovic, Sketching Clothoid Splines Using Shortest Paths, Comput. Graph. Forum (2010) 655–664.
URL `http://www.mit.edu/~ibaran/curves/`

[24] M. Brezak, I. Petrovic, Path smoothing using clothoids for differential drive mobile robots., International Federation of Automatic Control 18 (2011) 1133–1138.
URL `http://dx.doi.org/10.3182/20110828-6-IT-1002.02944`

[25] E. Bissé, M. Bentounes, E. Boukas, Optimal path generation for a simulated autonomous mobile robot, Autonomous Robots 2 (1) (1995) 11–27.
URL `http://dx.doi.org/10.1007/BF00735436`

[26] T. Fraichard, A. Scheuer, From Reeds and Shepp's to continuous-curvature paths, Robotics, IEEE Transactions on 20 (6) (2004) 1025–1035.
URL `http://dx.doi.org/10.1109/TRO.2004.833789`

[27] N. Montes, M. Mora, J. Tornero, Trajectory Generation based on Rational Bézier Curves as Clothoids, in: Intelligent Vehicles Symposium, 2007 IEEE, 2007, pp. 505–510.
URL `http://dx.doi.org/10.1109/IVS.2007.4290165`

[28] Y. Liu, H. Yang, W. Wang, Reconstructing B-spline Curves from Point Clouds–A Tangential Flow Approach Using Least Squares Minimization, in: Shape Modeling and Applications, 2005 International Conference, 2005, pp. 4–12.
URL `http://dx.doi.org/10.1109/SMI.2005.39`

[29] M. Stojmenovic, A. Nayak, Measuring Linearity of Ordered Point Sets, in: D. Mery, L. Rueda (Eds.), Advances in Image and Video Technology, Vol. 4872 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2007, pp. 274–288.
URL `http://dx.doi.org/10.1007/978-3-540-77129-6_26`

[30] J. Bentley, T. Ottmann, Algorithms for Reporting and Counting Geometric Intersections, Computers, IEEE Transactions on C-28 (9) (1979) 643–647.
URL `http://dx.doi.org/10.1109/TC.1979.1675432`

[31] D. K. Wilde, Computing clothoid segments for trajectory generation., in: IROS, 2009, pp. 2440–2445.
URL `http://dx.doi.org/10.1109/IROS.2009.5354700`