Optimal Path Synthesis for Automated Guided Vehicles

Preliminary Research

Reijer Idema

2005

TU Delft
Julianalaan 134
2628 BL Delft

FROG Navigation Systems
Krommewetering 21
3543 AP Utrecht

Optimal Path Synthesis for Automated Guided Vehicles

Preliminary Research

Author:
Reijer Idema

Supervisors:
prof.dr.ir. P. Wesseling *(TU Delft)*
dr.ir. Kees Vuik *(TU Delft)*
ir. Patrick H.F. Segeren *(FROG)*
dr.ir. René Jager *(FROG)*

February 15, 2005

Preface

This is the report of the preliminary research for the degree of Master of Science for the study Applied Mathematics, faculty of Electrical Engineering, Mathematics and Computer Science of the Delft University of Technology. The graduation work is done in the unit of Numerical Analysis, and takes nine months of work. The first three months focus on problem definition, study of literature, and planning the research for the next six months. Of that period of preliminary research, this is the report.

The research project is being carried out at FROG Navigation Systems. FROG is a manufacturer of Automated Guided Vehicles. They have developed a multitude of vehicles that transport products within factories of companies like Sony and General Motors, but also have an automatically driven bus for public transport driving in the center of Eindhoven. For more information see their website `www.frog.nl`.

Contents

List of Symbols

Chapter 1

Introduction

There is an ever rising demand for Automated Guided Vehicles. In industrial settings the use of robots and robot vehicles has been common practice for years already, but also in public transport and entertainment AGVs are started to be used. FROG Navigation Systems is a company that works to meet this demand.

The name FROG is short for Free Ranging on grid. FROG builds vehicles with a magnetic sensor that can read its position from a grid of magnets in the work area. The control software FROG manufactures, is able to freely move the vehicle over this grid.

Most AGV applications are highly repetitive, i.e., the vehicle has to do the same task over and over again. Therefore it is very important that the task is executed as efficient as possible. Also, people who have to work in the same area as the AGV prefer that the vehicle follows the same predictable route every time. For these reasons, usually a fixed path is used by the vehicle.

FROG has developed a path design tool, in which a vehicle path can be drawn by a human designer using NURBS curves. This implementation works, but leaves much to be desired. For a human designer it is not so hard to draw a path that avoids walls and other obstacles, but vehicle constraints like how fast a wheel can turn, can only be gotten right by trial and error. Also, there is no guarantee that a designed path is optimal, or even a good path. Therefore designing a vehicle path is a very tedious and time consuming task, and the quality of the result is generally unknown.

Mathematical enhancements could help to make vehicle path design easier. In this report we will take a mathematical look a the problem of generating good controls for an AGV. The goal is to develop a tool that can find the optimal controls for a vehicle, given its task. This report start with a formal problem definition in Chapter 2.

Chapter 3 will focus on the practical problem. Some difficulties are illustrated by examples, and ideas are offered to overcome them. Also tools are discussed that can be used to solve the vehicle control problem. The most important tool for now are NURBS curves, which are used to describe a path. Chapter 4 gives an overview of the relevant NURBS theory.

Finally, Chapter 5 uses the results from the earlier chapters to draw a general planning for the remaining six months of research. Due to the complexity of this problem, we will focus on a planning that generates useful tools for FROG at each intermediate state. Ultimately our goal is to combine these tools into an all-in-one vehicle control solution, that can generate an optimal path for any FROG vehicle given its task.

Chapter 2

Problem Definition

The automated guided vehicle problem we address in this report is as easy to explain as it is hard to solve.

Suppose an AGV has to perform an action in the world.
Find the best control input for the AGV to achieve the action. (2.1)

For many of the problems that fall into this category some control input can be found that achieves the goal, without too much effort of man and machine. However proving that such a control is the best, or even a good one, is usually much more complex.

To be able to find an optimal solution for this problem, we will have to translate the textual problem (2.1) to a mathematical problem. We will need a mathematical translation of all the keywords: vehicle, world, task and control. This chapter describes the mathematical framework we will use to model the problem.

The role we assume is that of the driver of the vehicle. We know the action we want the vehicle to perform and translate it to actions on the controls of the vehicle. The vehicle engine and other mechanics then translate the controls to actions. This translation may seem straightforward to any experienced driver, but mathematically it is not. It is therefore important to consider the controls and the actions of the vehicle separately.

We will call the vehicle controls and their possibilities and nonpossibilities the internal world, and the possible actions of the vehicle with their restrictions the external world. Accordingly, we split the mathematical model in an internal and an external component. The internal model mathematically describes the controls that operate the vehicle. The external model contains all relevant information about its position and shape, and the world around the vehicle.

Obviously the internal and external model component are linked by the mechanics of the vehicle. To link them mathematically, we will define a process function that describes the influence of internal vehicle controls on the external world. Further we will need a cost function, that measures how good or bad a control input is to achieve the desired action.

In the following sections we define our framework for an arbitrary vehicle world V, which contains all relevant information about the state of the vehicle and its environment. See Figure 2.1 on page 6 for an overview.

2.1 External Model

This section introduces the terms external state, external state function and external constraint, and describes their use in the external model of our vehicle problem.

The external state is a description of the position and orientation of the vehicle, but also contains things like the status of a manipulator, e.g., the fork of a fork-lift. All aspects of the vehicle that play a role in the interaction with its environment should be captured.

Definition 2.1.1. *The external state space (or placement space) is a space P that uniquely determines the external state of the vehicle.*

The external state space describes all possible placements of the vehicle. Therefore any action of the vehicle can be represented by a function that describes the external state of the vehicle at each point in time.

Definition 2.1.2. *An external state function (or placement function, or vehicle action) is a function $p : [0, \infty) \to P$ that describes the external state of the vehicle for the time period $[0, \infty)$.*

Definition 2.1.3. *The external state function space (or placement function space, or action space) is the space $\mathcal{P} = \{p : [0, \infty) \to P\}$ of all possible actions of the vehicle.*

Depending on the vehicle and its environment not all external state functions of the vehicle may be possible. There are all kinds of constraints on the actions of the vehicle due to its environment. A common constraint is that the vehicle needs a collision free path to its destination, i.e., for all $t \in [0, \infty)$ the vehicle in external state $p(t)$ may not collide with any obstacles in its environment. Another common example is a restriction on the maximum velocity depending on the position of the vehicle. A vehicle can be allowed to move much faster on a freeway than on a busy factory floor for example.

Definition 2.1.4. *An external state function constraint (or placement function constraint, or action constraint) is a function* $K_{ext} : \mathcal{P} \to \{0, 1\}$ *that describes a constraint on the external state function for the vehicle by the external world. If* $K_{ext}(p) = 1$ *then the action* p *is allowed, and if* $K_{ext}(p) = 0$ *then the action* p *is not allowed under the constraint* K_{ext}*.*

Since action constraints only depend on the external world we will also call them external constraints.

We will denote the space of all actions allowed under K_{ext} by:

$$K_{ext}(\mathcal{P}) := \{p \in \mathcal{P} \mid K_{ext}(p) = 1\}.$$

This concludes the description of the mathematical framework of the external model.

2.2 Internal Model

This section introduces the terms internal state, internal state function and internal constraint, and describes their use in the internal model of our vehicle problem.

The internal state is a description of the state of the controls that operate the vehicle. All aspects of the operational control of the vehicle needed for the efficient execution of its task should be captured in the control.

Definition 2.2.1. *The internal state space (or control space) is a space* Q *that uniquely determines the the control input for the vehicle.*

The internal state space describes all possible control inputs of the vehicle. Therefore any operation sequence for the vehicle can be represented by a function that describes the control input for the vehicle at each point in time.

Definition 2.2.2. *An internal state function (or control function, or vehicle operator) is a function* $q : [0, \infty) \to Q$ *that describes the internal state of the vehicle for the time period* $[0, \infty)$*.*

Definition 2.2.3. *The internal state function space (or control function space, or operator space) is the space* $\mathcal{Q} = \{q : [0, \infty) \to Q\}$ *of all possible operators for the vehicle.*

Depending on the vehicle not all control functions for the vehicle may be possible. Possible constraints on the control function include restrictions on the operation speed of the gas pedal, on the angular velocity of the steering wheel, and restrictions on the use of manipulators.

Definition 2.2.4. *An internal state function constraint (or control function constraint, or operator constraint) is a function* $K_{int} : \mathcal{Q} \to \{0, 1\}$ *that describes a constraint on the internal state function for the vehicle by the internal world. If* $K_{int}(q) = 1$ *then the operator* q *is allowed, and if* $K_{int}(q) = 0$ *then the operator* q *is not allowed under the constraint* K_{int}*.*

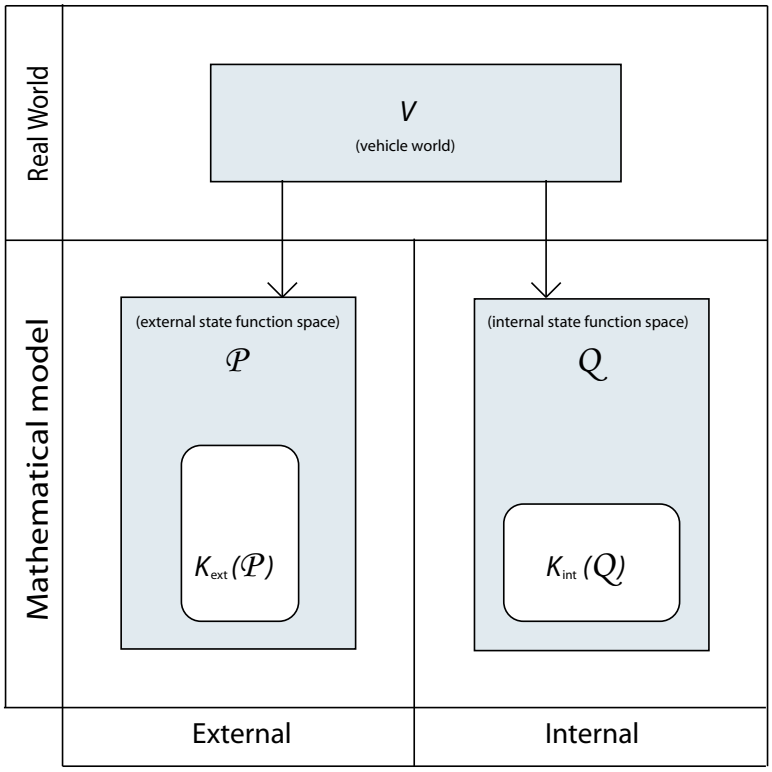
Since operator constraints only depend on the internal world we will also call them internal constraints.

We will denote the space of all operators allowed under K_{int} by:

$$K_{int}(\mathcal{Q}) := \{q \in \mathcal{Q} \mid K_{int}(q) = 1\}.$$

This concludes the description of the mathematical framework of the internal model. Figure 2.1 below provides a visual representation of the external and internal model.

Figure 2.1: Model overview



2.3 Engine Model

This section introduces the process function. The process function models the vehicle mechanics, mapping a control input to the resulting vehicle action. This action usually depends on the initial external state $p_0 \in P$ of the vehicle. This is because we cannot know where we are going if we did not know where we started, but also because the action may depend on the position in the work space. For example, with the same energy input the vehicle may go uphill slower than it would go downhill. For convenience reasons we adopt the metaphor engine function for the process function.

Definition 2.3.1. *The process function, or engine function, is a function $E : (\mathcal{Q} \times P) \to \mathcal{P}$ that, given an initial external state $p_0 \in P$, maps a control function $q \in \mathcal{Q}$ to the action it will induce on the vehicle:*

$$p = E(q, p_0), \text{ with } p(0) = p_0.$$

Note that in theory this mapping is exact. However in practice our description of the world can never be so complete that the exact action a certain control input will cause is known. If it is important to model this fact, then the engine function can be made stochastic.

Further note that no control or state function constraints have been taken into account in the definition of the engine function. We will use this generality in the definition of the actual optimization problem below.

2.4 Optimal Vehicle Control

Suppose we have modeled a vehicle world V and want the vehicle to perform a certain task. Tasks can range from a factory robot transporting boxes of eggs to a storage room without breaking them, to an entertainment ride driving around for 5 minutes while shaking up the passengers really good. This section describes the optimization problem that has to be solved to find the best action to perform the desired task.

Definition 2.4.1. *A task function for a vehicle is a function $T : \mathcal{P} \to \{0, 1\}$ with the following property:*

$$T(p) = 1 \quad \text{if } p \text{ performs the action,}$$
$$T(p) = 0 \quad \text{otherwise.}$$

We will often identify a task by its task function, writing "the task T" for "the task with task function T".

Given a task T, we define the space of actions that perform the task, and the space of control functions that make the vehicle perform the task:

$$\begin{aligned} T(\mathcal{P}) &:= \{p \in \mathcal{P} \mid T(p) = 1\} \\ T(\mathcal{Q}, p_0) &:= \{q \in \mathcal{Q} \mid T(E(q, p_0)) = 1\} \end{aligned}$$

Again, for generality purposes, no state or control function constraints have been taken into account. Although the elements of the space are said to perform the desired task, it is possible that they are not practically executable due to internal or external constraints.

To be able to tell which control function is the best to perform the desired task, we need some measure that quantifies how good a certain control function is.

Definition 2.4.2. *A cost function for task T, for a vehicle with initial external state $p_0 \in P$, is a function $C_q^T : (\mathcal{Q} \times P) \to \mathbb{R}$ with the properties:*

$$\begin{aligned} C_q^T(q, p_0) &= \infty && \text{if the control function } q \notin T(\mathcal{Q}, p_0), \\ C_q^T(q_1, p_0) &\leq C_q^T(q_2, p_0) && \text{if } q_1 \text{ not worse than } q_2,\ q_1, q_2 \in T(\mathcal{Q}, p_0). \end{aligned}$$

Typical quantities assessed by a cost function are the stress on the vehicle hardware, the stress on passengers or other load and the energy consumption during the execution of the task.

We can now formulate the optimization problem, that has as optimal solution the best control function for a vehicle to perform its task. If there is no feasible solution, or if all feasible solutions have an infinite large cost, then it is not possible for the vehicle to perform the task under the modeled conditions.

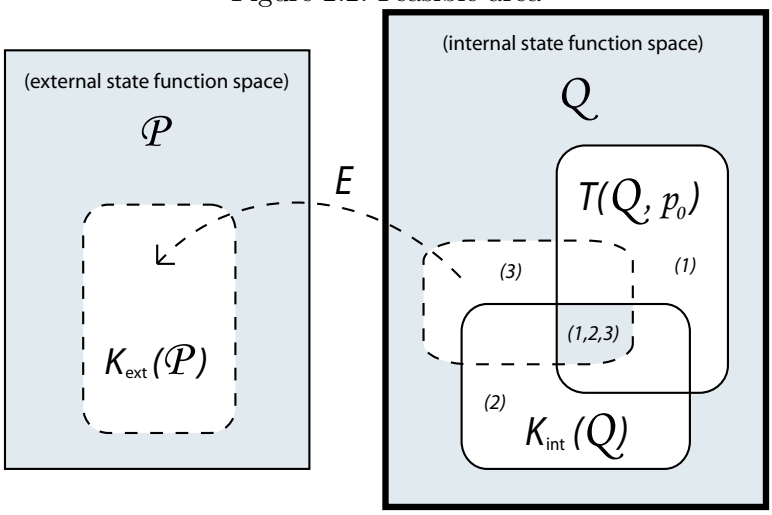
Definition 2.4.3. *Assume a vehicle world V has been modeled for a vehicle with initial external state $p_0 \in P$, and let T be the task function for the task we want the vehicle to perform. Let K_{int} and K_{ext} denote the union of all internal and external constraints respectively, and let C_q^T be the cost function we want to minimize. Then the optimal control function is the function q that solves the following problem:*

$$\begin{aligned} \min_{q \in \mathcal{Q}} \quad & C_q^T(q, p_0) \\ s.t. \quad & q \in T(\mathcal{Q}, p_0) \\ & q \in K_{int}(\mathcal{Q}) \\ & E(q, p_0) \in K_{ext}(\mathcal{P}) \end{aligned} \tag{2.2}$$

Note that we have not explicitly modeled an initial internal state $q_0 \in Q$ for the vehicle. When needed we can simply add the condition $q(0) = q_0$ to the optimization problem above, or we can model the initial internal state as an internal constraint. Of course p_0 and q_0 should be coherent, if there is a relation between them in the vehicle world.

Figure 2.2 below shows the feasible area of the optimization problem. The sets of $q \in \mathcal{Q}$ satisfying the demands $q \in T(\mathcal{Q}, p_0)$, $q \in K_{int}(\mathcal{Q})$ and $E(q, p_0) \in K_{ext}(\mathcal{P})$, are denoted in the figure by the areas (1), (2) and (3) respectively. The feasible area is the set of control functions q that satisfies all three demands. This is the gray area marked (1,2,3) in the figure.

Figure 2.2: Feasible area



Our objective is to find the control function q in area (1,2,3) with the lowest cost. This function q will be the best control input for the vehicle, to perform the action T.

Chapter 3

Practical Implementation

This chapter presents some general concepts and ideas for the practical implementation of the vehicle control model presented in the previous chapter. To get more insight into the problem at hand, we start with building a model for a very simple fork-lift truck problem. After that we will use the issues arisen in the fork-lift truck example, to identify general implementation issues and provide methods to deal with them. Finally, we will introduce some tools we intend to use for building the model and solving the problem.

3.1 Fork-lift Truck Example

In this section we will explore the vehicle control model, using a simple fork-lift truck example. The truck can drive forward and backward at a fixed velocity, steer, and put its fork-lift up or down. Its work space is a rectangular flat floor, and its task is to pick up some crates.

3.1.1 External Model

The external state of the fork-lift truck can be fully specified by its position and orientation, and the state of the fork-lift. Since the work space is two dimensional we only need two coordinates x and y for the position and one rotation ϕ for the orientation of the fork-lift truck. The state of the fork-lift we will denote by z.

Already we face a choice. We can choose $x, y \in \mathbb{R}$ and define the confinement of the work space in the external constraints totally, or we can translate some, or maybe even all, of the work space geometry to bounds on the x and y coordinate. Usually these bounds will also be a function of ϕ.

Which of these methods is the best, may depend on the exact optimization problem formulation and the solving method used. It should also be taken into consideration that, if we bound the values of x and y, the external state space may have to be redefined if the work space of the vehicle chances.

So from a flexibility viewpoint it is best to keep the external state space as general as possible, unless we know beforehand that this will give problems when solving the optimization problem. Therefore in this example we will choose $x, y \in \mathbb{R}$ and $\phi \in [0, 2\pi)$, where (x, y) is the position of a reference point on the vehicle and ϕ denotes the counterclockwise angle with the x-axis. The state of the fork-lift can be described with a boolean: $z = 0$ when the fork-lift is down, and $z = 1$ when it is up.

A value (x, y, ϕ, z) uniquely determines the external state. So we can define the external state space of the vehicle as

$$P = \mathbb{R} \times \mathbb{R} \times [0, 2\pi) \times \{0, 1\},$$

and an action is thus a function

$$(x(t), y(t), \phi(t), z(t)) : [0, \infty) \to P.$$

The external constraints is where we really incorporate the work space of the vehicle. We will assume that the maximum velocity of the fork-lift truck is not so high that it needs limiting by a constraint.

This leaves us with the task to build a constraint that ensures the vehicle only drives where it is allowed to. The complexity of this task heavily depends on the shape of the vehicle and the work area. In the case of a polygonal work space and a circular vehicle, it is fairly easy to build an explicit formulation of the allowed work space. As the shape of the vehicle and that of the work area become more complicated however, incorporating this constraint becomes quite challenging.

Even in our simple example we face a difficult task. Of course, the fork-lift truck generally is not allowed to drive through the crates. However, to pick them up it will have to shove its fork-lift under the crates. This can only be done from a certain side, with the fork-lift in the down position. Modeling this into a constraint is far from straightforward.

Also, as soon as the crates are picked up by the fork-lift, their status changes from “an obstacle that can be picked up” to “a load on the fork-lift truck”. In our model of the external state we have left out a variable describing if there is a load or not. However, if we want to model constraints that depend on the presence of a load on the fork-lift, we will need this variable. For example, we may want to prevent the vehicle truck from lowering its fork-lift while driving with a load.

3.1.2 Internal Model

The internal state space is a model of the state of the controls of the vehicle. Our fork-lift truck has three controls: a fork-lift control f, a gas pedal g, and a steering wheel ψ. We presume that the gas pedal is a switch with three modes: forward, backward and halt. Further the steering wheel can be turned 180 degrees clockwise and 180 degrees counterclockwise, and the fork-lift control is a switch with the modes down and up.

An intuitive model of the above description of the controls is to choose $f \in \{0, 1\}$ for the fork-lift control (0 for down and 1 for up), $g \in \{-1, 0, 1\}$ for the gas pedal (1 for forward, 0 for halt, and -1 for backward), and $\psi \in [-\pi, \pi]$ for the position of the steering wheel.

Now a value (f, g, ψ) uniquely determines the internal state of the fork-lift truck. So we can define the internal state space of the vehicle as

$$Q = \{0, 1\} \times \{-1, 0, 1\} \times [-\pi, \pi],$$

and a control function is thus a function

$$(f(t), g(t), \psi(t)) : [0, \infty) \to Q.$$

Since the gas and the fork-lift control are switches, we do not need constraints on their control. We assume that each switch can be set to the desired position in an instant. The actual reaction of the external state of the vehicle to a change in one of the switches, is the responsibility of the engine function. This includes delays, accelerating to the desired velocity and lifting the fork to the desired height.

Unlike the discrete switches, the steering wheel is a continuous control that cannot be jerked from one position to another in an instant. We can model this constraint by limiting the angular velocity of the steering wheel, i.e., $\psi'(t) \leq c$ for some c representing the maximum angular velocity.

There is one more remark to make. So far we have only considered constraints that come from how the controls let themselves be operated. However, it is likely there are also constraints due to the system that has to operate the controls. Almost any control system is discrete, i.e., it applies control changes at certain time intervals. Even if these intervals are very short, this may have serious consequences for the practical control of the vehicle.

3.1.3 Engine Model

The engine model describes how the controls of the vehicle affect its actions. Let us start with the fork-lift control, because it is the easiest. The external

state of the fork-lift is directly connected to the internal control switch. If the switch is set to 0, the fork-lift of our simplified example will immediately be in the down position, and if it is set to 1, the fork-lift will be instantly in the up position. Thus we have:

$$z(t) = f(t).$$

The other components of the external state function are far less straightforward. We start with a description of the orientation $\psi(t)$, because we will need it for the positional components.

Let $\alpha(t)$ be the direction in which the vehicle is driving at time t. We will assume that the change in this direction is some function D of the position of the steering wheel ψ:

$$\frac{d\alpha}{dt}(t) = D(\psi(t)).$$

Then we have:

$$\alpha(t) = \alpha_0 + \int_0^t D(\psi(u))\, du.$$

For simplicity reasons we will further assume that the orientation ϕ of the vehicle is the same as the direction in which the vehicle is driving:

$$\psi(t) = \alpha(t).$$

Then the position of the vehicle at time t is given by:

$$\begin{aligned} x(t) &= x_0 + \mu \int_0^t g(u) \cos\alpha(u)\, du, \\ y(t) &= y_0 + \mu \int_0^t g(u) \sin\alpha(u)\, du, \end{aligned}$$

where μ is the absolute velocity of the vehicle when driving.

3.1.4 Optimal Control

A cost function for the fork-lift truck problem would probably be some function of the time spend on the task T, and the energy needed to perform the task. Let us assume that the energy consumption of the vehicle is constant, so it only depends on the time spend on the task. Then the cost function would be:

$$C_q^T(q, p_0) = \tau(q),$$

where $\tau(q)$ is the time needed to perform the task.

Filling in the cost function and the results of the previous sections in problem (2.2), gives the optimization problem for the fork-lift truck problem. Even for such a simple problem, the modeling as well as the optimization problem is quite complex, and the solution is far from straightforward.

3.2 Implementation Issues

The main conclusion to be drawn from the fork-lift truck example, is that our problem is generally very complex. Even this very simple example, gives rise to some challenging modeling work.

In practical cases the optimization problem (2.2) will not be in any standard form, and no algorithm exists that solves the general problem. To solve it optimally, we will therefore have to treat each case individually with a specialized method, or we have to simplify the model to a form we can generally solve. Note that in the latter case, although we would be able to solve the model optimally, the model itself will usually be less exact.

Another option would be to use an approximate solution method, like a local search algorithm. In this case we do not have to further simplify our model, but we will have to discretize the solution space and the solution provided by such a method may be nonoptimial. This could of course also be combined with simplifications to the model.

In the next few sections we present two simplification concepts for the vehicle control problem. Also, we will introduce a few possible tools for the modeling and solving of the problem, with their advantages and disadvantages.

3.2.1 Task Splitting

One of the most used methods to simplify a complex problem, is by splitting it up into smaller problems. Ideal is a splitting that does not change the optimal solution space of the problem. But the real power of this type of simplification lies in the separation of components that are loosely coupled, i.e., of which the decoupling has no major effect on the optimal solution.

If at any point of the vehicle task the internal and external state are completely defined, we can split the task into a task to reach that state and a task from that state on forward. If a certain subtask could be executed before or after the defined splitting point, we could evaluate both options independently to find the best option. This does not change the solution space of the problem.

Another very interesting simplification is the following. In most practical

applications, or at least in the ones we are pursuing, manipulator actions can be defined quite exhaustively. As a consequence there is hardly any room for optimization in the manipulator actions. In this case we can take the manipulator components out of the external state description, leaving just the positional and orientation component.

For example, take the fork-lift truck problem. Picking up the crates is a very precisely defined action. We could define this action by hand, and build a simpler model to optimize the action of driving to the starting point of the action to pick up the crates, neglecting the fork-lift.

Due to the above observations, when modeling the external state of the vehicle, we will focus our attention on the path of the vehicle, i.e., its position and orientation through time.

3.2.2 Constraint Projection

Part of the complexity of our problem is caused by the fact that we have to worry about the internal and external state function simultaneously. In the problem definition (2.2) we use the engine function E to describe all constraints as a function of the internal state function q. But this function E is in practice very complex, consequently making the optimization problem (2.2) very complex.

To eliminate this difficulty we could try to project the external constraints onto the internal state function space, i.e., try to formulate constraints on the control function q that ensure that the induced action $p = E(q, p_0)$ satisfies the original external constraints. Or we could project the internal constraints onto the external state function space.

The first option, projecting the external constraints onto the control function space, seems the more natural if we think of the vehicle problem solver as a mathematical version of the human driver it replaces. For a human driver interprets the external world, and converts what he sees into rules like "go left here, or we will hit a wall". These kind of rules are in fact external constraints projected onto the control function space. However, we prefer the second option, projecting the internal constraints to the placement function space, for several reasons.

The external constraints contain, amongst others, the constraints describing the work area of the vehicle. The work area can be very large and very complex, with moving obstacles for example. Also it may radically change from time to time.

The internal constraints of the vehicle problem are less complex than the external constraints. They are driven by the natural laws of the controls and

actuators of the vehicle. Projecting the internal constraints to the external state space is generally much easier than projecting all obstacles in the work area to the control function space. An example is the translation of constraints on the steering controls to bounds on, and continuity requirements for, the derivatives of the placement function.

We want to allow interaction between the software that works on the AGV problem, and its users. Users visualize and think in placement functions, much more than in control functions. Therefore the software will have to work with the placement function p externally, and it is a more natural choice for the software to focus on the placement function p internally too. Of course, this is only possible if a mapping from a placement function to a control function exists. We will call this mapping the driver, because it has the same role as a traditional human driver, i.e., translating a desired course to operations on the controls of the vehicle.

Definition 3.2.1. *The controller function, or driver function, is a function* $D : (\mathcal{P} \times Q) \rightarrow \mathcal{Q}$ *that, given an initial internal state* $q_0 \in Q$*, maps a placement function* $p \in \mathcal{P}$ *to the control function that executes that action:*

$$q = D(p, q_0), \text{ with } q(0) = q_0.$$

Note that $E(D(p, q_0), p(0)) = p$ and $D(E(q, p_0), q(0)) = q$. Thus we can view the driver and the engine functions as the inverse of each other.

We will denote the space of all external state functions allowed under the internal constraints $K_{int}(\mathcal{Q}, p_0)$ by:

$$K_{int}(\mathcal{P}, q_0) := \{p \in \mathcal{P} \mid D(p, q_0) \in K_{int}(\mathcal{Q})\}.$$

So the space $K_{int}(\mathcal{P}, q_0)$ is the projection of the internal constraints to the external state space, and we can define the space of placement functions allowed under all constraints by:

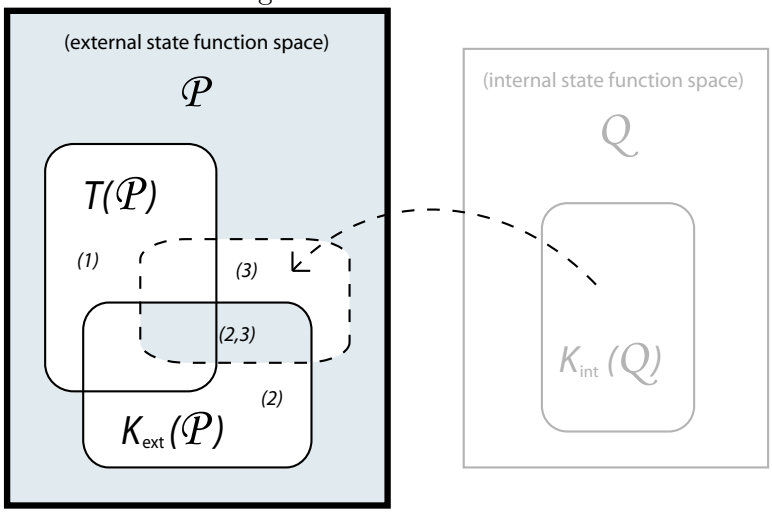
$$K(\mathcal{P}, q_0) := K_{ext}(\mathcal{P}) \cap K_{int}(\mathcal{P}, q_0).$$

Figure 3.1 below shows the projection of the internal constraints onto the external state function space. The space $K_{int}(\mathcal{P}, q_0)$ of all external state functions allowed under the internal constraints, is the area marked by (3). The gray area marked by (2,3) is the space $K(\mathcal{P}, q_0)$ of placement functions allowed under all constraints.

Now, if we also remodel the cost function to be a function of an external state function, we can redefine the vehicle problem in a form that only uses the action space $\mathcal{P}$ and an initial internal condition $q_0 \in Q$. So we define the cost function $C_p^T : (\mathcal{P} \times Q) \rightarrow \mathbb{R}$ by:

$$C_p^T(p, q_0) = C_q^T(D(p, q_0), p(0)).$$

Figure 3.1: Feasible area



Definition 3.2.2. *Assume a vehicle world V has been modeled for a vehicle with initial internal state $q_0 \in Q$, and let T be the task function for the task we want the vehicle to perform. Let K denote the union of all external constraints and all externally projected internal constraints, and let C_p^T be the cost function we want to minimize. Then the optimal placement function is the function p that solves the following problem:*

$$\begin{aligned} \min_{p \in \mathcal{P}} \quad & C_p^T(p, q_0) \\ s.t. \quad & p \in T(\mathcal{P}) \\ & p \in K(\mathcal{P}, q_0) \end{aligned} \tag{3.1}$$

For a certain vehicle world, sets of internal and external constraints, task, and cost function, this optimization problem is equivalent to problem (2.2) in the following manner. If $\bar{p}$ is an optimal solution of (3.1) for the initial internal condition q_0, then $\bar{q} = D(\bar{p}, q_0)$ is an optimal solution of (2.2) for the initial external condition $\bar{p}(0)$.

Of course having to project the internal constraints using the driver function is no real simplification of our problem. The idea is, however, to model the internal constraints as constraints on the placement function directly. In practice this will usually take some heuristics and approximations.

So, instead of modeling the internal constraints $K_{int}(\mathcal{Q})$ and then calculating the external projection $K_{int}(\mathcal{P}, q_0)$, we will model a set of constraints $\tilde{K}_{int}(\mathcal{P}, q_0) \subseteq K_{int}(\mathcal{P}, q_0)$ directly. Similarly, we will also model the cost function as a function of the vehicle action p from the start.

3.2.3 Tools

We have presented a mathematical model for the vehicle problem (2.1), and simplifications to make it easier to work with and less hard to solve. In this section we give an overview of existing tools we will be going to use, and mention some tools that would be useful to develop.

In the previous sections 3.2.1 and 3.2.2 we showed how to simplify the problem to a path problem in the external state function space, eliminating vehicle manipulators, like the fork-lift of a fork-lift truck, and the internal state function space from the problem. Therefore we will focus on tools for the path problem.

To be able to efficiently work with vehicle paths, we will need a way to describe them, i.e., we need a convenient form for the external state function. From the start of this project it was the intention to use NURBS curves. For an overview of the NURBS curve theory see Chapter 4.

These Non-Uniform Rational B-Splines have several very useful features, that make them an excellent choice for this problem. For example the fact that any order of continuity is easily achieved by choosing an appropriate degree of the NURBS curve. The local support property is also very important, since it implies that we can make local changes to the curve without upsetting a distant part of the curve that was already correct.

Of course there are also disadvantages to using NURBS. The mathematical description of a NURBS curve is quite complex, and a useful operation like calculating if two curves intersect is not straightforward. But in our opinion the advantages more than outweigh the disadvantages.

NURBS curves are very adept at describing the position of a point in space through time, but the path of a vehicle has other dimensions than just the position of a point. The most important is the orientation of the vehicle. But there may be more, like another orientation dimension in case of a articulated bus. We will have to investigate if these can best be added as an extra NURBS dimension, or that they should be added independently of the NURBS curve that describes the position of a point on the vehicle.

Having a mathematical model and a description of the external state function, we need a solver to find the best, or at least a very good, path. Due to the complexity of the problem we take special interest in noncomplete search methods. Local search methods seem to us like a natural way to handle the problem. The before mentioned local support property of NURBS suits the needs of local search methods. For information on the subject of local search methods, see [AL97] and [HS04].

A search method needs two tools to be able to search through the solution

space. First, it must be able to obtain an initial solution. We intend to start with a rough polynomial approximation of the path the vehicle should follow, specified by a user or calculated by planning software for example. The second tool needed by a search method is a way to generate variations on a solution.

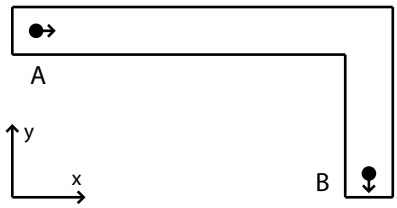
A path variation is only useful if the placement function p performs the task T and meets all constraints, i.e., $p \in T(\mathcal{P})$ and $p \in K(\mathcal{P}, q_0)$. When a lot of variations are generated that do not have these properties, the search method will be very inefficient. Therefore we intend to explore the possibilities of task and constraint preserving operations, i.e., operations that, when used on a placement function that performs the task and meets the constraints, give another placement function with these properties.

For a lot of search methods it is important to have a good set of heuristics, to make the search more efficient. Take for example the fact that, for a lot of vehicles with steering wheels, straight lines and parts of circles are efficient paths since they only require steering action between two such segments. Generating path variations with a lot of these segments, might direct a search method to a good solution faster. Another obvious heuristic is the fact that, in a lot of cases, a shorter path provides a better solution.

3.2.4 Example

To demonstrate the discussed implementation issues, we will treat a very simple example. A vehicle has to drive from point A to B on a road with one corner, as shown in Figure 3.2 below. We suppose that possible other task components have been split out, to leave us with just the problem of finding an optimal path.

Figure 3.2: Work Area



To avoid unnecessary complication at this point, we assume that the vehicle is a point mass, and that its work area is a flat floor. Further we suppose that the vehicle can only drive at a fixed velocity μ, or be at a halt, and that the orientation of the vehicle is always the same as the direction in which

it drives, i.e., the front always points in the direction of movement. So it is not possible for the vehicle to move sideways.

The steering wheel can be turned to any angle $\psi \in [-\psi_{max}, \psi_{max}]$ instantly, and the effect of a fixed steering angle is that the vehicle corners along a circle. The radius r of the cornering circle is small if ψ is large, yielding a sharp corner, and if $\psi = 0$ then r is infinitely large, yielding a straight trajectory of the vehicle. So $r = \rho(|\psi|)$ for some monotonic decreasing function ρ on the interval $(0, \psi_{max}]$, with $\rho > 0$ for all ψ. Therefore the sharpest corner the vehicle can take has radius $r_{min} = \rho(\psi_{max})$.

Figure 3.3: Steering Model

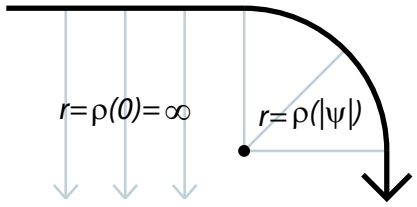


Figure 3.3 shows the path of the modeled vehicle, when the steering angle is fixed at $\psi = 0$ at the beginning, and then changes to some $\psi \neq 0$. Note that, even though ψ is discontinuous in time, the vehicle path is still continuous.

From this general description we will build our mathematical model. We start with the external model. Since the vehicle is a point mass, and the work area is a flat floor, we can model its external state as a 2-dimensional position (x, y). Therefore the external state space is

$$P = \mathbb{R} \times \mathbb{R},$$

and a placement function is a function

$$(x(t), y(t)) : [0, \infty) \to P.$$

The orientation of the vehicle is the direction of the vector $(x'(t), y'(t))$, and the velocity is equal to the length of that vector.

The only constraint on the placement function is that (x, y) should always be in the available space, i.e., the vehicle is not allowed to drive through walls. Requirements on the starting and ending position and orientation are all part of the task function.

The internal state of the vehicle is specified by its throttle position f and steering wheel angle ψ. So the internal state space is

$$Q = \{0, 1\} \times [-\psi_{max}, \psi_{max}],$$

and a control function is a function

$$(f(t), \psi(t)) : [0, \infty) \to Q.$$

There are no further internal constraints.

The next step is to project the internal constraints onto the external state space. Note that the fact that there are no formally defined internal constraints, does not mean that we can skip the process of constraint projection. The shape of the internal state space itself also has an effect on what actions are feasible, and should therefore be projected to the external state space.

The external effect of the velocity restriction $v \in \{0, \mu\}$ is that the vehicle cannot jump to another position, i.e., $x(t)$ and $y(t)$ should be continuous functions. Changing the position of the steering wheel only changes the radius of the circle the vehicle moves along. It does not allow the vehicle to start driving in a different direction than it was facing (see Figure 3.3). So $x'(t)$ and $y'(t)$ should also be continuous functions.

We assume a setting in which the only important factor is how fast the vehicle performs its task. So the cost of an action $p = (x(t), y(t))$ is the time needed to drive from A to B. It is obvious that the cost for a path becomes higher when the vehicle stops for some time before B is reached. Therefore we can assume that the vehicle does not stop along the way, and take the length of the vehicle path as a measure for the cost of that path.

This effectively splits the velocity out of the problem. As a consequence we don't have to worry about the length of $(x'(t), y'(t))$, which means that we can take a different parameterization of the placement function is we want.

Because we do not know the time needed to execute a task beforehand, we had to define external state functions on the infinite time interval $[0, \infty)$. But when we are free to choose a parameterization, independent of execution time, we can parameterize the path on the finite interval $[0, 1]$. So we can choose the placement function to be:

$$p(u) = (x(u), y(u)) : [0, 1] \to P.$$

The projection of the internal state space then results in the requirement that $p(u)$ and $p'(u)$ are continuous, i.e., $p \in C^1$.

The task T to drive from A to B can now be described by the constraints

$$p(0) = A,\ p'(0) = (1, 0)\ ,\ p(1) = B \text{ and } p'(1) = (0, -1),$$

and the cost of an action p can be measured by

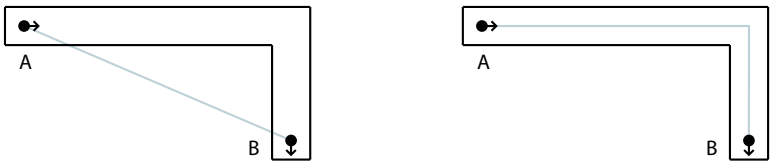
$$C_p^T(p) = \int_0^1 \left|\left|p'(u)\right|\right| du.$$

We will now treat some placement functions p for this problem that go from A to B, and discuss their relation to the model developed above.

First we consider the two paths in Figure 3.4 below. The left path is inspired by the fact that the shortest distance between two points is a straight line. If the vehicle would be able to follow this path it would therefore be the optimal path. However it is obvious that this is not a feasible path. The path satisfies the internal constraints but the external constraints are violated because the vehicle would have to drive through walls. Also, this path does not even perform the task T properly, because the starting and ending orientation of the vehicle are not correct.

The path p on the right in Figure 3.4 does perform the task properly. Also it does not drive through walls, so the external constraints are met. However, at the corner $p'(u)$ is not continuous, so the internal constraints are violated. It is impossible to steer around this corner with the vehicle.

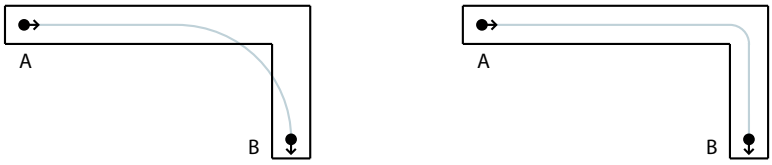
Figure 3.4: Straight Paths



Next we consider the two paths in Figure 3.5 below. Both resemble the path on the right of Figure 3.4, but now the corners are rounded. The path on the left has obviously been rounded too much, because the vehicle would have to drive through walls. Supposing that the radius r of the corner of the path on the right is not too small for the vehicle to perform, i.e., $r \geq r_{min}$, that path is a feasible path for our vehicle problem.

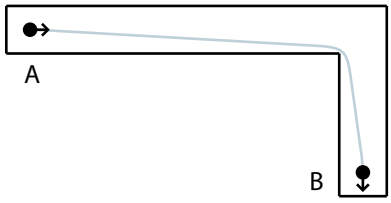
Inspired by this result, we observe the following simple heuristic to attempt to build a feasible path for this problem. Start with a polygonal approximation of the path, and then round all the corners to satisfy $r \geq r_{min}$, hoping that the rounded corner will not go through a wall.

Figure 3.5: Rounded Paths



Now that we have a feasible path, we would like to generate feasible variations, in attempt to find a shorter feasible path. The optimal path will look something like the path in Figure 3.5. Note that a little cornering at A and B is necessary to get the starting and ending orientation right.

Figure 3.6: Optimal Path



Chapter 4

NURBS

In the previous chapters we build a model for the AGV control problem, and presented methods to reduce the practical case to an optimal path problem. As mentioned in Section 3.2.3, we need a way to represent the vehicle path.

NURBS curves are a type of splines that are widely used in computer aided design, because of their flexibility and geometric properties, combined with the fact that NURBS algorithms are fast and numerically stable. Due to the complexity of our problem, a solver will have to rely heavily on heuristics. Thus the solver can be seen as smart automated shape designer, for which it is a natural choice to work with NURBS curves.

This chapter treats the NURBS matter that we feel is important to the AGV problem. The book [PT97] is used as a reference for more detailed information and algorithms which we do not want to include in this overview. This book is widely regarded as a standard work on the matter. Due to its focus on modeling and design, [CRE01] may also be of interest. For a more mathematical treatment of splines in general, we refer the reader to [dB01].

4.1 Preliminaries

The path of a vehicle is a multi-dimensional curve. Such a curve can be described in different form, e.g., parametric and implicit. We will use parametric curves. This section provides some knowledge needed for the understanding of this chapter.

Definition 4.1.1. *A parametric polynomial of degree n and dimension d is a function*

$$\mathbf{C}(u) = (x_1(u), \ldots, x_d(u)) \quad , \quad u \in [a, b],$$

with $x_i(u)$ polynomials, and the highest degree of these $x_i(u)$ equal to n.

Polynomials can be denoted in many forms. We will present two forms that are vital to the NURBS matter that we will treat here, the power basis curve form and the Bézier curve form.

Definition 4.1.2. *Given* $\mathbf{a}_0, \ldots, \mathbf{a}_n \in \mathbb{R}^d$, $\mathbf{a}_n \neq \mathbf{0}$, *a parametric polynomial of degree* n *and dimension* d *in power basis form is a function*

$$\mathbf{C}(u) = \sum_{i=0}^{n} \mathbf{a}_i u^i \ , \quad u \in [a, b] \, .$$

The vectors $\mathbf{a}_i$ *are called the coefficients, and* $\mathbf{a}_n$ *the leading coefficient.*

Definition 4.1.3. *Let a set of control points* $P = \{\mathbf{P}_0, \ldots, \mathbf{P}_n\}$ *be given, with* $n \geq 0$ *and* $\mathbf{P}_i \in \mathbb{R}^d$. *A Bézier curve, given the control points* P, *is a curve*

$$\mathbf{C}(u; P) = \sum_{i=0}^{n} \mathbf{P}_i B_{i,n}(u) \ , \quad u \in [0, 1]$$

where the Bernstein basis functions $B_{i,n}$ *are given by*

$$B_{i,n}(u) = \frac{n!}{i!\,(n-i)!} u^i (1-u)^{n-i} \ , \quad u \in [0, 1] \, .$$

Of course polynomials cannot describe all curves. Circles and ellipses are among the curves a polynomial cannot precisely represent. All conic curve can, however, be described with rational functions, i.e., functions that are the ratio of two polynomials.

Definition 4.1.4. *Let* $n \geq 0$, *and let a set of weights* $W = \{w_0, \ldots, w_n\}$, *with* $w_i > 0$, *and a set of control points* $P = \{\mathbf{P}_0, \ldots, \mathbf{P}_n\}$ *be given, with* $\mathbf{P}_i \in \mathbb{R}^d$. *A rational Bézier curve, given the control points* P *and the weights* W, *is a curve*

$$\mathbf{C}(u; P, W) = \sum_{i=0}^{n} \mathbf{P}_i R_{i,n}(u) \ , \quad u \in [0, 1]$$

where the basis functions $R_{i,n}$ *are given by*

$$R_{i,n}(u; W) = \frac{B_{i,n}(u)\, w_i}{\sum_{j=0}^{n} B_{j,n}(u)\, w_j} \ , \quad u \in [0, 1] \, .$$

Rational curves of dimension d can be represented by a nonrational curve of dimension $d+1$, by using homogeneous coordinates. Let us define the following perspective map $\mathcal{H}$:

$$\mathcal{H}(x_1, \ldots, x_d, w) := \begin{cases} \left(\frac{x_1}{w}, \ldots, \frac{x_d}{w}\right) & , \ w \neq 0 \\ \text{direction } (x_1, \ldots, x_d) & , \ w = 0 \end{cases}$$

Given a rational Bézier curve $\mathbf{C}(u; P, W)$ with $\mathbf{P}_i = (x_{i1}, \ldots, x_{id})$, define the $d+1$ dimensional points $\mathbf{P}_i^w := (w_i x_{i1}, \ldots, w_i x_{id}, w_i)$. Then the curve

$$\mathbf{C}^w(u; P^w) := \sum_{i=0}^{n} \mathbf{P}_i^w B_{i,n}(u) \quad , \quad u \in [0,1]$$

is a nonrational Bézier curve, and

$$\mathcal{H}(\mathbf{C}^w(u; P^w)) = \mathbf{C}(u; P, W).$$

4.2 B-Spline Curves

Complex shapes are hard to describe with a single polynomial. A very high degree polynomial is needed to accurately represent a complex curve. Piecewise polynomial functions are far better equipped to deal with with such curves. B-splines are a piecewise polynomial extension of Bézier curves. In this section we will give a definition of B-spline curves, and treat some of their properties that are important to our application.

Definition 4.2.1. *Let $m > 0$, and let a set $U = \{u_0, \ldots, u_m\}$ be given, with $u_i \in [a,b]$ nondecreasing and $u_0 = a$, $u_m = b$. We call U the knot vector, and u_i the knots. Let $0 \leq n < m$, and let a set of control points $P = \{\mathbf{P}_0, \ldots, \mathbf{P}_n\}$ be given, with $\mathbf{P}_i \in \mathbb{R}^d$. Define the degree p as:*

$$p := m - n - 1.$$

A B-spline, given the control points P and the knot vector U, is a curve

$$\mathbf{C}(u; P, U) = \sum_{i=0}^{n} \mathbf{P}_i N_{i,p}(u; U) \quad , \quad u \in [a,b]$$

where the basis functions $N_{i,p}(u; U)$, for $i \leq n$, are given by the recursive formula

$$\begin{aligned} N_{i,0}(u; U) &= 1_{[u_i, u_{i+1})} = \begin{cases} 1 & , \quad u \in [u_i, u_{i+1}) \\ 0 & , \quad \textit{otherwise} \end{cases} \\ N_{i,k}(u; U) &= \frac{u - u_i}{u_{i+k} - u_i} N_{i,k-1}(u; U) + \frac{u_{i+k+1} - u}{u_{i+k+1} - u_{i+1}} N_{i+1,k-1}(u; U). \end{aligned}$$

$N_{i,k}$ can contain the quotient $\frac{0}{0}$. This quotient is defined to be 0.

Unless there are multiple knot vectors or sets of control points involved, we will denote $\mathbf{C}(u; P, U)$ and $N_{i,k}(u; U)$ by $\mathbf{C}(u)$ and $N_{i,k}(u)$ respectively.

4.2.1 Knot Vector Properties

The knots $u_{p+1}, \ldots, u_{m-p-1}$ are called the internal knots. A B-spline is called uniform if all internal knots are equally spaced, i.e., if $u_{i+1} - u_i$ is equal for all $p \leq i \leq m - p - 1$.

The multiplicity of a knot is the number of times the value of that knot occurs in U. Note that these occurrences are necessarily consecutive. A B-spline is called nonperiodic if the first and last knot have multiplicity $p + 1$, i.e., if $u_0 = \ldots = u_p = a$ and $u_{m-p} = \ldots = u_m = b$. B-splines with a nonperiodic knot vector have the useful property that $\mathbf{C}(u_0) = \mathbf{P}_0$ and $\mathbf{C}(u_m) = \mathbf{P}_n$. From now on, we will assume knot vectors to be nonperiodic unless otherwise stated.

4.2.2 Basis Function Properties

The basis functions $N_{i,p}$ only depend on the degree p and knot vector U. More precisely, the shapes of the basis functions are determined entirely by p and the relative spacing between the knots. Translating and scaling the knot vector does not change the shape of the basis functions.

For any i and p, we have $N_{i,p}(u) = 0$ if $u \notin [u_i, u_{i+p+1})$. This is called the local support property. A direct consequence is that, in any knot span $[u_i, u_{i+1})$, at most $p + 1$ of the $N_{i,p}$ are nonzero, namely the functions $N_{j-p,p}, \ldots, N_{j,p}$.

For any i, we have $\sum_{j=1}^{n} N_{j,p}(u) = \sum_{j=i-p}^{i} N_{j,p}(u) = 1$ for all $u \in [u_i, u_{i+1})$. This is called the partition of unity property. Also, we have nonnegativity, i.e., $N_{i,p}(u) \geq 0$ for all i, p and u.

Each $N_{i,p}$ is polynomial on any knot span, hence all derivatives of the basis functions exist, and are therefore necessarily continuous, on the interior of a knot span. At a knot $N_{i,p}(u)$ is $p-k$ times continuously differentiable, where k is the multiplicity of the knot. Therefore, increasing the degree p increases continuity, and increasing the knot multiplicity k decreases continuity.

The k^{th} derivative of a basis function, for $k \leq p$, is given by:

$$N_{i,p}^{(k)} = \frac{p!}{(p-k)!} \sum_{j=0}^{k} a_{k,j} N_{i+j,p-k}$$

with:

$$\begin{aligned} a_{0,0} &= 1 \\ a_{k,0} &= \frac{a_{k-1,0}}{u_{i+p-k+1} - u_i} \\ a_{k,j} &= \frac{a_{k-1,j} - a_{k-1,j-1}}{u_{i+p+j-k+1} - u_{i+j}} \ , \quad j = 1, \ldots, k-1 \\ a_{k,k} &= \frac{-a_{k-1,k-1}}{u_{i+p+1} - u_{i+j}} \end{aligned}$$

Whenever a denominator with knot differences becomes 0, the quotient is defined to be 0. Obviously, all higher derivatives are zero.

For efficient algorithms to compute and evaluate basis functions and their derivatives, see §2.5 of [PT97].

4.2.3 B-spline Properties

The B-spline $\mathbf{C}(u)$ is a piecewise polynomial curve, for it is a linear combination of the basis functions $N_{i,p}$, which are polynomial on each knot span. For the same reason, $\mathbf{C}(u)$ is infin and at least $p - k$ times continuously differentiable at a knot with multiplicity k.

When the derivatives of the basis functions are known, the derivatives of the B-spline are easy to compute:

$$\mathbf{C}^{(k)}(u) = \sum_{i=0}^{n} \mathbf{P}_i N_{i,p}^{(k)}(u) \ , \quad u \in [a, b] \, .$$

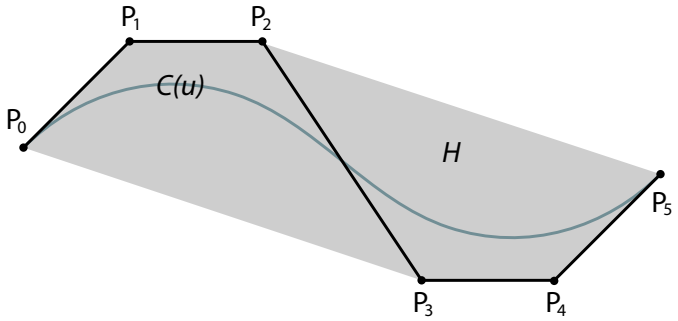
For an algorithm that evaluates a B-spline at a certain u see §3.2 of [PT97], and for algorithms that evaluate derivatives see §3.3 of that book.

As mentioned before, we have $\mathbf{C}(u_0) = \mathbf{P}_0$ and $\mathbf{C}(u_m) = \mathbf{P}_n$ (assuming that U is nonperiodic). Due to the local support property of the basis functions, moving a control point $\mathbf{P}_i$ modifies the curve $\mathbf{C}(u)$ only on the interval $[u_i, u_{i+p+1})$. This is called the local modification scheme. If the degree p increases, the support also increases, and the effect of changes in a control point will be less localized.

The control polygon of a B-spline is the polygon of all line segments between two consecutive control points, i.e., the control polygon is the collection of line segments $\{\mathbf{P}_0\mathbf{P}_1, \mathbf{P}_1\mathbf{P}_2, \ldots, \mathbf{P}_{n-1}\mathbf{P}_n\}$, where $\mathbf{P}_i\mathbf{P}_j$ denotes the line segment between $\mathbf{P}_i$ and $\mathbf{P}_j$. The convex hull of the control polygon is the set of all points H, for which two points on the control polygon exist, such that the line segment between these two points goes through H. See Figure 4.1 for a graphical illustration of the control polygon and its convex hull.

A B-spline curve is contained in the convex hull of its control polygon. Moreover, if $u \in [u_i, u_{i+1})$ with $i \in \{p, \ldots, m - p - 1\}$, then $\mathbf{C}(u)$ is in the

Figure 4.1: Control Polygon and Convex Hull



convex hull of the control points $\mathbf{P}_{i-p}, \ldots, \mathbf{P}_i$. This is known as the strong convex hull property. As a consequence of this property, if $\mathbf{P}_{i-p}, \ldots, \mathbf{P}_i$ with $i \in \{p, \ldots, m-p-1\}$ are collinear, then the curve $\mathbf{C}(u)$ will be a straight line segment for $u \in [u_i, u_{i+1})$.

Another property of B-splines is the variation diminishing property, which is the fact that no plane (or line in the 2-dimensional case) has more intersections with the curve than with the control polygon of the curve. An important consequence of this property is that, if a plane has no intersections with the control polygon, it also has no intersections with the curve.

4.3 NURBS Curves

A NURBS curve is a Non-Uniform Rational B-Spline curve, i.e., a B-spline on a nonuniform knot vector with rational basis functions.

Definition 4.3.1. *Let $m > 0$, and let a knot vector $U = \{u_0, \ldots, u_m\}$ be given, with $u_i \in [a, b]$ nondecreasing and $u_0 = a$, $u_m = b$. Let $0 \leq n < m$, and let a set of weights $W = \{w_0, \ldots, w_n\}$, with $w_i > 0$, and a set of control points $P = \{\mathbf{P}_0, \ldots, \mathbf{P}_n\}$ be given, with $\mathbf{P}_i \in \mathbb{R}^d$. Define the degree p as:*

$$p := m - n - 1.$$

A NURBS curve, given control points P, knot vector U and weights W, is a curve

$$\mathbf{C}(u; P, U, W) = \sum_{i=0}^{n} \mathbf{P}_i R_{i,p}(u; U, W) \ , \quad u \in [a, b] \ ,$$

where the rational basis functions $R_{i,p}$ are given by

$$R_{i,p}(u; U, W) = \frac{N_{i,p}(u; U)\, w_i}{\sum_{j=0}^{n} N_{j,p}(u; U)\, w_j} \ .$$

Unless multiple knot vectors, sets of control points or sets of weights are involved, we will denote $\mathbf{C}(u; P, U, W)$ and $R_{i,k}(u; U, W)$ by $\mathbf{C}(u)$ and $R_{i,k}(u)$ respectively.

NURBS curves are even more flexible than ordinary B-splines, because of the extra weight parameters. If all weights are equal, the curve is a B-spline. Circles, ellipses and hyperbolas are important examples of curves that cannot be represented by a polynomial function like a B-spline, but can be precisely described by a rational function like a NURBS curve.

4.3.1 Basis Function Properties

The rational basis functions $R_{i,p}$ have the local support, partition of unity and nonnegativity properties. Also, they are infinitely differentiable on the interior of a knot span, and $p-k$ times continuously differentiable in a knot with multiplicity k.

4.3.2 NURBS Properties

NURBS curves are piecewise rational curves. They are infinitely differentiable on the interior of a knot span, and at least $p-k$ times continuously differentiable in a knot with multiplicity k. Also, for a nonperiodic knot vector we have $\mathbf{C}(u_0) = \mathbf{P}_0$ and $\mathbf{C}(u_m) = \mathbf{P}_n$, and a nonperiodic NURBS curve without interior knots is a rational Bézier curve.

Like nonrational B-splines, NURBS have the strong convex hull property, the variation diminishing property, and the local modification scheme property. The local modification scheme is even extended to include changes in the weights w_i, i.e., moving a control point $\mathbf{P}_i$, or changing a weight w_i, modifies the curve $\mathbf{C}(u)$ only on the interval $[u_i, u_{i+p+1})$.

4.3.3 Homogeneous Representation

We can represent a NURBS curve of dimension d as a nonrational B-spline of dimension $d+1$, using homogeneous coordinates, just like we did with rational Bézier curves in Section 4.1.

Given a NURBS curve $\mathbf{C}(u; P, U, W)$ with $\mathbf{P}_i = (x_{i1}, \ldots, x_{id})$, define the $d+1$ dimensional points $\mathbf{P}_i^w := (w_i x_{i1}, \ldots, w_i x_{id}, w_i)$. Then the curve

$$\mathbf{C}^w(u; P^w, U) := \sum_{i=0}^{n} \mathbf{P}_i^w N_{i,p}(u; U) \quad , \quad u \in [0,1]$$

is a nonrational B-spline, and

$$\mathcal{H}\left(\mathbf{C}^w\left(u; P^w, U\right)\right) = \mathbf{C}\left(u; P, U, W\right).$$

We can use the homogeneous representation $\mathbf{C}^w$ to make it easier to work with NURBS. All basic operations on NURBS can be done by applying the operation on this nonrational B-spline representation, with no, or some minor work, added to preserve the operation under the mapping. For example, the derivatives of a NURBS curve $\mathbf{C}$ can be expressed as a function of the derivatives of the corresponding B-spline $\mathbf{C}^w$. Therefore, we will focus on operations on B-splines for the remainder of this chapter.

4.4 B-Spline Operations

4.4.1 Elementary Operations

Knot Insertion

Knot insertion is the process of adding a knot to the knot vector, and consequently, a control point to the set of control points. All curves that could be constructed with the old knot vector by changing P, are possible with the new knot vector, and more. Therefore, knot insertion can be used to increase flexibility in shape control. Other important applications are curve splitting, and the evaluation of points and derivative values on curves.

Let a knot vector $U = \{u_0, \ldots, u_m\}$ be given. Suppose we want to insert a knot v in the k^{th} knot span of U. Then the new knot vector is given by $V = \{v_0 = u_0, \ldots, v_k = u_k, v_{k+1} = v, v_{k+2} = u_{k+1}, \ldots, v_{m+1} = u_m\}$. Given a curve $\mathbf{C}\left(u; P, U\right)$, we want to find a set of control points Q, such that

$$\mathbf{C}\left(u; Q, V\right) = \mathbf{C}\left(u; P, U\right) \ , \quad u \in [0, 1].$$

There always exists a unique Q that accomplishes this, see §5.2 of [PT97].

By adding a knot a number of times, such that its multiplicity becomes equal to the degree p, we can split a curve at that knot. Since the evaluation of the curve and its derivatives is easy at endpoints, we can also use this splitting for evaluation purposes.

Often, we want to insert many knots at once. This is called knot refinement. It can be accomplished by inserting knots one by one, but more efficient algorithms exist. Such an algorithm is presented in §5.3 of [PT97].

Knot refinement can be used to make the effects of modifications on the control point more local. But it can also be used to decompose a curve into its polynomial (Bézier) segments.

Knot Removal

Knot removal is the reverse process of knot insertion. It can be used to obtain a more compact form of the curve representation, e.g., to remove redundant knots after operations which involved knot insertions or after combining curve segments to one new curve.

Suppose we have a curve $\mathbf{C}(u; P, U)$. Let $\tilde{u}$ be an internal knot that has multiplicity k in the knot vector U. We say that $\tilde{u}$ is $0 \leq r \leq k$ times removable, if a set of control points $Q = \{\mathbf{Q}_0, \ldots, \mathbf{Q}_{n-r}\}$ exists, such that

$$\mathbf{C}(u; Q, V) = \mathbf{C}(u; P, U) \ , \quad u \in [0, 1]$$

where V is the knot vector U from which $\tilde{u}$ is removed r times.

In computational practice we have to allow deviations in the curve within a small tolerance, due to round-off errors. For more information, and a knot removal algorithm, see §5.4 of [PT97].

Degree Elevation

Given a curve $\mathbf{C}_p(u; P, U)$ of degree p, we want to find a knot vector V and set of control points Q, such that the curve $\mathbf{C}_{p+1}(u; Q, V)$ is of degree $p+1$ and it is geometrically and parametrically equal to $\mathbf{C}_p(u; P, U)$, i.e.,

$$\mathbf{C}_{p+1}(u; Q, V) = \mathbf{C}_p(u; P, U) \ , \quad u \in [0, 1] \, .$$

This process is called degree elevation. The space of all NURBS curves of degree $p+1$ is a vector space that includes all curves of degree p. Therefore, degree elevation is always possible.

An application of degree elevation is making two B-spline curves of the same degree, in order to be able to combine them into one single curve. For examples and a degree elevation algorithm, see §5.5 op [PT97].

Degree Reduction

Degree reduction is the inverse process of degree elevation. Given a curve $\mathbf{C}_p(u; P, U)$ of degree p, we want to find a knot vector V and set of control points Q, such that the curve $\mathbf{C}_{p-1}(u; Q, V)$ is of degree $p-1$ and it is geometrically and parametrically equal to $\mathbf{C}_p(u; P, U)$, i.e.,

$$\mathbf{C}_{p-1}(u; Q, V) = \mathbf{C}_p(u; P, U) \ , \quad u \in [0, 1] \, .$$

Like knot removal, degree reduction is not always possible, and in computational practice we have to allow deviations in the curve within a small tolerance, due to round-off errors. An obvious application of degree reduction, is the approximation of a high-degree curve with a low-degree curve. For examples and a degree reduction algorithm, see §5.6 op [PT97].

4.4.2 Advanced Operations

Point Inversion and Point Projection

Given a point $\mathbf{P}$ on the curve $\mathbf{C}(u)$, point inversion is the problem of finding the parameter $\bar{u}$, such that $\mathbf{C}(\bar{u}) = \mathbf{P}$. Point inversion cannot be solved analytically for $p > 4$, and the computational solution for $p = 3$ and $p = 4$ is not without problems.

Given a point $\mathbf{P}$, not necessarily on the curve $\mathbf{C}(u)$, point projection is the problem of finding the parameter $\bar{u}$ for which the distance between $\mathbf{C}(\bar{u})$ and $\mathbf{P}$ is minimal. Point projection can be numerically solved by using Newton iterations, see §6.1 of [PT97]. We can also use this method to solve the point inversion problem, accepting $\bar{u}$ as the point inversion parameter if the distance between $\mathbf{C}(\bar{u})$ and $\mathbf{P}$ is within a certain error tolerance.

Transformations

Transformations on B-splines include translations, rotations, scalings, shears and reflections. All these transformations can be performed by applying the operation on the control points.

Reparameterization

Let $f : [c, d] \to \mathbb{R}$ be a differentiable function with $f'(s) > 0$ for all s. Then the composition of $\mathbf{C}(u)$ and $f(u)$ is a reparameterization of $\mathbf{C}(u)$:

$$\mathbf{C}(s) := \mathbf{C}(f(s)).$$

Methods to calculate the reparameterization, for f being a polynomial or rational function, can be found in §6.4 of [PT97].

The concepts used for reparameterization can also be used for a function $f : [c, d] \to \mathbb{R}$ with $f'(s) < 0$ for all s. This fact can be used to reverse the direction of a curve, as is described in §6.5 of the same book.

Conversion to and from Piecewise Power Basis Forms

Since B-splines are piecewise polynomial, they can also be represented in piecewise power basis form. Details on the conversion of one form into the other, and conversion algorithms, can be found in §6.6 of [PT97].

4.4.3 Shape Modification Operations

In this section we will describe some shaping tools that allow us to make local modifications to a curve, in a way that is much more intuitive than the mathematical tools described above that. These shaping operations are

very useful for the implementation of heuristics on path variations. A more detailed survey of these tools can be found in Chapter 11.1 of [PT97].

Two direct methods to modify the shape of a curve are control point repositioning and weight modification. There are multiple ways to implement control point repositioning. We can specify new coordinates for a control point directly, or we could change the position of a point on the curve and calculate the control points that accomplish this modification.

Weight modification is a useful tool to push part of a curve to, or pull it from, a control point. Increasing the weight w_i locally moves curve points $\mathbf{C}(u)$, along the line through that point $\mathbf{C}(u)$ and $\mathbf{P}_i$, in the direction of $\mathbf{P}_i$. Even small pushes and pulls often require substantial weight modification. Because widely varying weights can give poor parameterizations, often leading to unexpected problems, in practice weight modification should be kept small.

Besides these low level shaping tools, a range of high level tools exist. Some important examples are warping, flattening, bending and constraint-based curve shaping. Warping locally deforms a curve using a shape function as a model for the deformation. Flattening introduces a straight line segment and bending bends a part of the curve. Constraint based curve shaping forces positional and derivative constraints on the curve at certain parameter values. This tool has a direct application to our vehicle path problem. We will need some form of it to impose the constraints on the vehicle path.

4.5 Curve Fitting

Curve fitting techniques synthesize a curve according to certain requirements, like a set of points that have to be on the curve and derivative and continuity requirements. Interpolation and approximation are two fitting methods. Interpolation tries to generate a curve that matches the input data exactly. Approximation builds a curve that fits the requirements within a certain error tolerance.

Curve fitting methods can be either global or local. A global fitting method sets up a problem and solves it to get the entire curve at once. Local changes to the input data may therefore have a global effect on the generated curve. Local fitting methods construct the curve segment by segment. Local changes to the input data will only have a local effect on the curve, and these methods can deal with local anomalies better. However, continuity requirements can be problematic at the segment boundaries.

Curve fitting methods are not unique. A lot of methods exists to solve curve fitting problems, many of the heuristic. A selection of such methods can be found in Chapter 9 of [PT97].

Chapter 5

Research Planning

We have built a general mathematical problem description for the vehicle control problem (2.1), and briefly explored what tools we need for a practical implementation of a solver for this problem. The next step is the development of these tools and, ultimately, the implementation of a solver. This chapter points out the important subtasks in the development of this solver, and plans in what order they can best be researched.

We assume that all vehicle actions that are not related to the path it drives, have been split out of the problem as described in Section 3.2.1. Thus we can focus on the simple vehicle task of driving from a point A to a point B.

We will start our research with a simple point mass vehicle, so we can focus on the general basics. We assume that the vehicle follows a NURBS path with its orientation always in the direction of movement. Further we will disconnect the velocity from the path entirely, as we did in the example in Section 3.2.4. This allows more freedom in the parameterization of the path curve.

When the research looks promising we will add a body to the vehicle model, which complicates the process of satisfying the external constraints. Adding a body is absolutely vital for any practical application.

Another addition would be to consider vehicles that can crab, i.e., vehicles that can move in a different direction than they are facing. To implement crabbing, an extra dimension in the path curve is needed to describe the orientation of the vehicle. There are all kinds of other vehicles conceivable that require higher path dimensions, e.g., articulated buses like the Phileas FROG has running in Eindhoven.

5.1 Internal Constraint Research

First we plan to work on the projection of internal constraints. We will need to develop a method to synthesize a NURBS curve that satisfies these constraints, or to transform an existing curve that does not satisfy the constraints to a related curve that does. Further we intend to develop operations on NURBS curves that preserve the internal constraints. These are necessary to be able to generate variations on a curve, without having to worry about the internal constraints.

The current implementation of the synthesis of a NURBS path at FROG, is a NURBS design tool layered over a CAD drawing of the work area. The user has to modify control points and weights to design a good path. The hardest problem is to satisfy the internal constraints with this method. A tool that can generate a path that satisfies the internal constraints, and provides operations on this path that preserve this property, can therefore be of great use even before it is implemented in an automated problem solver.

Previous research of FROG has already uncovered a lot of information about the consequences of internal constraints on the path a vehicle can follow. It seems that there is a general mathematical description of these projected internal constraints, that is valid for the better part of the vehicle models currently at use. This would mean that if we are able to develop NURBS tools that can work with this general form, they could be used for most of the FROG vehicles without major adjustments.

5.2 External Constraint Research

The next step is to develop a method to incorporate external constraints. The main focus of this research will be on the calculation if a path is collision free. Other common external constraints, like speed limitations, will be much easier to implement, especially in the simple case in which we have disconnected the velocity from the path.

For a pointmass vehicle we only need a method to identify intersections of a NURBS path with the boundaries of the work area. But for a vehicle with a body we also have to be able to calculate the hull curve for a certain path, and intersect it with the environment. With the hull curve we mean the curve that separates the space in which no part of the vehicle will come, from the space off all points that are occupied by the vehicle at some time during the execution of the path.

5.3 Cost and Heuristics Research

Research is needed to find a cost function that gives a good classification of good and not so good paths. FROG already uses a value s, which is the distance covered by a strategically chosen point on the vehicle body, or a point outside the body with a fictional connection to it. For a good choice of such a point, s can be interpreted as a measure of the energy needed to lead the vehicle along the desired path. To be able to use this value as a cost function, we will have to find a method to calculate it for any given path.

It is very important to develop a good cost function before we start researching possible heuristics. Since the heuristics have to describe ideas of what could make a path better, they are highly dependent on the chosen cost function. A first step in the development of heuristics would be to identify the most cost efficient way to take a corner.

5.4 Solver Algorithm Research

When all the needed components have been researched and developed, we will have to devise a solver algorithm. The solver will have to combine the developed tools and ideas to find an optimal, or a very good, path.

Local search methods seem a very promising way to attack the AGV problem, using the above described tools and ideas. When we have a better insight in the exact problem structure and the solution requirement, we will have to research some choices of solvers, and test implementations to see if they can solve the problem in practice.

5.5 Conclusion

We have set out a planning for the research needed to solve the vehicle control problem (2.1). As always in research, it is very hard to estimate how much time each step will consume, or even if we will be able to solve the problem fully. Therefore it is very important to start with a simple problem, and extend only when we have successfully solved this problem.

The important thing to note about our research planning, is that each individual step provides a valuable tool in the design of good vehicle paths. They could be implemented in the current solution, in which the user designs the path totally, to radically simplify the work of this user. Therefore, the research will not be wasted if, for some reason, it is not possible to develop an all-in vehicle control solution.

Bibliography

[AL97] Emile Aarts and Jan Karel Lenstra, editors. *Local search in Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Chichester, 1997.

[CRE01] Elaine Cohen, Richard F. Riesenfeld, and Gershon Elber. *Geometric Modeling with Splines: An Introduction*. A K Peters, Natick, 2001.

[dB01] Carl de Boor. *A Practical Guide to Splines*. Springer-Verlag, New York, revised edition, 2001.

[HS04] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, San Francisco, first edition, 2004.

[PT97] Les Piegl and Wayne Tiller. *The NURBS Book*. Springer-Verlag, Berlin, second edition, 1997.