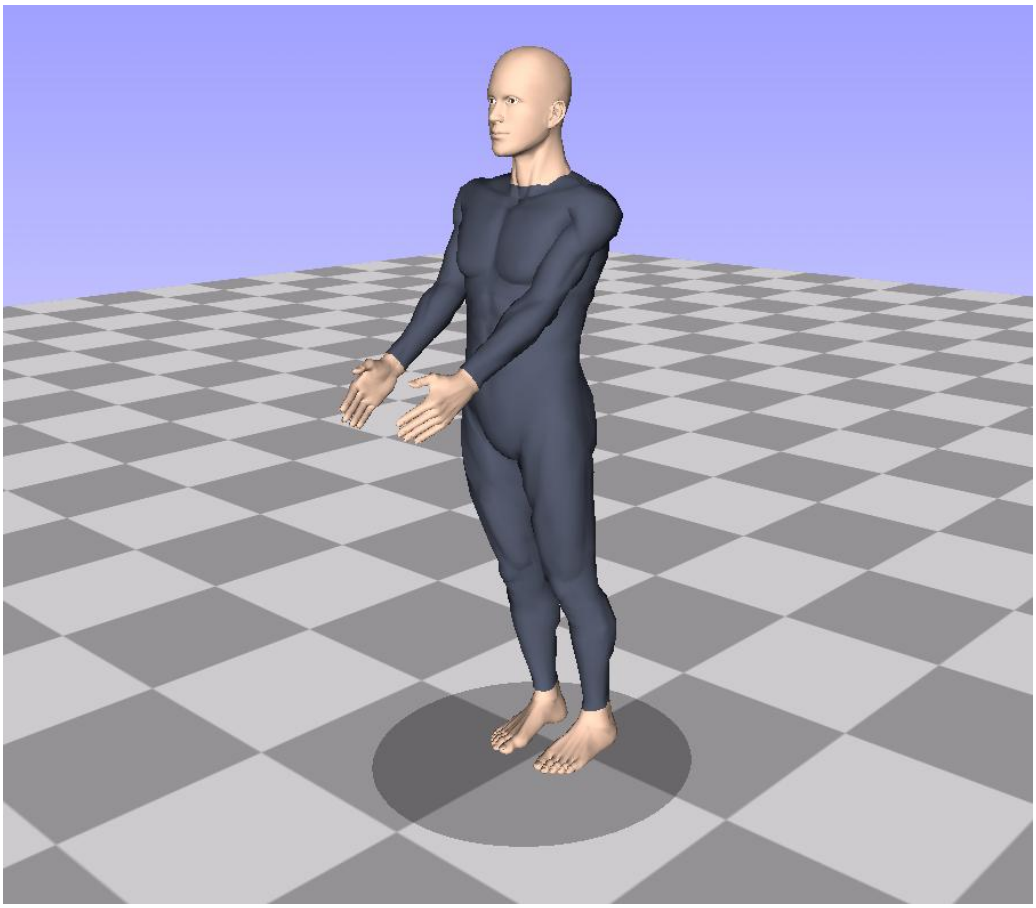
CHALMERS



Virtual Manikin Controller

Calculating the movement of a human model

Master's thesis in Complex Adaptive Systems

DANIEL GLEESON

Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2012
Master's thesis 2012:30

MASTER'S THESIS IN COMPLEX ADAPTIVE SYSTEMS

Virtual Manikin Controller

Calculating the movement of a human model

DANIEL GLEESON

Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2012

Virtual Manikin Controller
Calculating the movement of a human model
DANIEL GLEESON

Master's thesis 2012:30
ISSN 1652-8557
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
The cover shows the manikin - the human model - in the visualisation environment used by the project Intelligently Moving Manikin (IMMA) at Fraunhofer-Chalmers Research Centre for Industrial Mathematics (FCC).

Chalmers Reproservice
Gothenburg, Sweden 2012

Virtual Manikin Controller
Calculating the movement of a human model
Master's thesis in Complex Adaptive Systems
DANIEL GLEESON
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

Simulation of assembly line industrial robots has long been an area of great research interest. At Fraunhofer-Chalmers Research Centre for Industrial Mathematics (FCC) a main objective has been to automatically calculate valid motions - solving problems of inverse kinematics, path planning and collision avoidance. In order to include human assembly in the simulation environment the manikin project Intelligently Moving Manikin (IMMA) has been developed.

The IMMA project is able to calculate valid poses where problems such as balance, collision avoidance and ergonomics has been taken into account. This master thesis improves the manikin motion generation by limiting the velocity and acceleration, using the calculated poses as initial values, and stating the problem as a numerical optimal control problem. The control signal is optimised with respect to time and energy and is at the same time required to fulfil conditions on positions and limits of angular values, angular velocity and angular acceleration. All variables included in the problem formulation are discretised using low order Galerkin approximations and the nonlinear optimisation makes use of the open source interior point optimisation software Ipopt. The result is a smooth and efficient manikin movement which is displayed using the IMMA visualisation environment. The implementation is as general as possible in order to provide an optimisation framework where the objective function, numerical integration method and the constraints are all easily interchangeable.

Keywords: Manikin, Motion generation, Optimal Control, Optimisation

Acknowledgements

I would like to thank Staffan Björkenstam, who has been my advisor at FCC, for our discussions and for his help, guidance and support. My thanks also goes out to my examiner Krister Wolff for his input on my work and writing.

I would also like to thank all the people at FCC who have supported me, including Niclas Delfs for his guidance and Johan Carlson for his dedication to the project. They have also - together with Robert Bohlin, Daniel Segerdahl and Domenico Spensieri - come with great suggestions and discussed the project thoroughly during my intermediate presentations.

Finally, for their constant support and for their contributions to the report, I would like to thank Sofia Gunnarsson for creating most of the figures and my parents Ann and John Gleeson for their grammatical input.

Nomenclature

Ball-constraints	Spatial constraints specifying the maximal allowed deviation from a fixed target point.
Bang-off-bang solution	An optimal solution that only switches between three discrete values; max, min and zero.
Forward Kinematics	Finding the positions and rotations of all rigid bodies of a model, given all joint values.
Inverse Kinematics	The problem of finding joint values that match a TCP to a specified target.
KKT conditions	Necesary conditions for a solution of a non linear problem to be optimal.
Manikin	A simulated model of a human being.
Tool Center Point	The cartesian (x,y,z)-coordinates for the target points of interest, i.e. the hands of the manikin.

Abbreviations

FCC	Fraunhofer-Chalmers Research Centre for Industrial Mathematics
FEM	Finite Element Method
IMMA	Intelligently Moving Manikin (FCC project)
IPOPT	Interior Point OPTimizer
IPS	Industrial Path Solutions (FCC software)
KKT	Karush-Kuhn-Tucker (see KKT conditions above)
NLP	NonLinear Programming, synonym for nonlinear optimisation
ODE	Ordinary Differential Equation
TCP	Tool Center Point (see Tool Center Point above)

Contents

1 Introduction

In industrial production there is a great need to simulate the manufacturing process. Simulating and optimising the production can for example speed up assembly times, simplify logistics planning or detect faulty designs before the product is actually constructed. At Fraunhofer-Chalmers Research Centre for Industrial Mathematics (FCC) simulations of industrial robots is a research area of great interest. When simulating an industrial robot a few main areas of interest are path planning, collision avoidance, time optimisation and motion generation. The robot arm has to be able to move from one point to another, without running into fixed or moving objects and preferably do so in a smooth and efficient manner. At FCC the main focus has been laid on developing tools that automatically solves the path-planning problems that arise in industrial production. The developed software goes under the name Industrial Path Solutions (IPS) [1].

Expanding this area of industrial production, FCC has also developed a simulation environment for human assembly. The human model, or manikin, is developed to calculate collision free and ergonomically correct working positions. This is to ensure that the assembly is not only possible but that it also can be done in an efficient and ergonomically correct way. It is not only FCC that is involved in the manikin project, which goes under the name Intelligently Moving Manikin (IMMA). It is developed together with academia at Chalmers, Lund and Skövde as well as a number of industrial parters such as Volvo and Scania.

When modelling humans many of the problems are similar to the ones faced when simulating industrial robots. Since the model is built up in the same way as the robot, with rigid links connected by joints, the movement of the manikin is modelled in very much the same way as the robot. The ergonomic conditions are enforced just as viable working positions are checked for the robot and balance is maintained by requiring all forces and moments to cancel each other out.

There are however a few differences between simulating a robot and a human being. The manikin has typically a larger number of degrees of freedom. Not only are there a larger number of joints, the manikin can also move around in space. This is modelled by the root of the manikin, which is placed in the lower back, being linked to translational and rotational joints that position it according to a fixed reference frame.

It is a sequence of these working positions, or poses that set the target positions that the manikin should follow. This master thesis aims to describe and control the motion of the manikin through these target positions. In the thesis the possibility to use numerical optimal control methods to solve this type of problem will also be investigated.

1.1 Purpose

The master thesis aims at improving motion generation for manikins in the FCC software Industrial Path Solutions (IPS).

1.2 Problem Setup

A sketch of the manikin can be found in figure 1.2.1. The figure shows to the left how the manikin is built up by rigid links connected by joints and to the right a sketch of how a simplified example of this model can be represented by vectors and joint values. The model is simply a chain, or more precisely a tree, of connected rigid bodies and each connection is defined by a single parameter. The parameter can either be an angle to define a rotational joint or a length that defines a translational joint. For a single connection between two rigid bodies the two terms parent and child will be used. The parent is the rigid body that is closer to the stem of the tree while the children are closer to the branches. In a tree structure a rigid body can have multiple children, but only one parent.

In the left figure we have the two angles θ_1 and θ_2 that along with the two rotational axes k_1 and k_2 uniquely specify the position of the two vectors. In this simple example there are only two joints with their corresponding vectors but the manikin model is made up of about 160 of these joints, both rotational and translational. Six of these joints - three rotations and three translations - connect the lower back of the manikin to the origin. These joints allow the manikin to move around

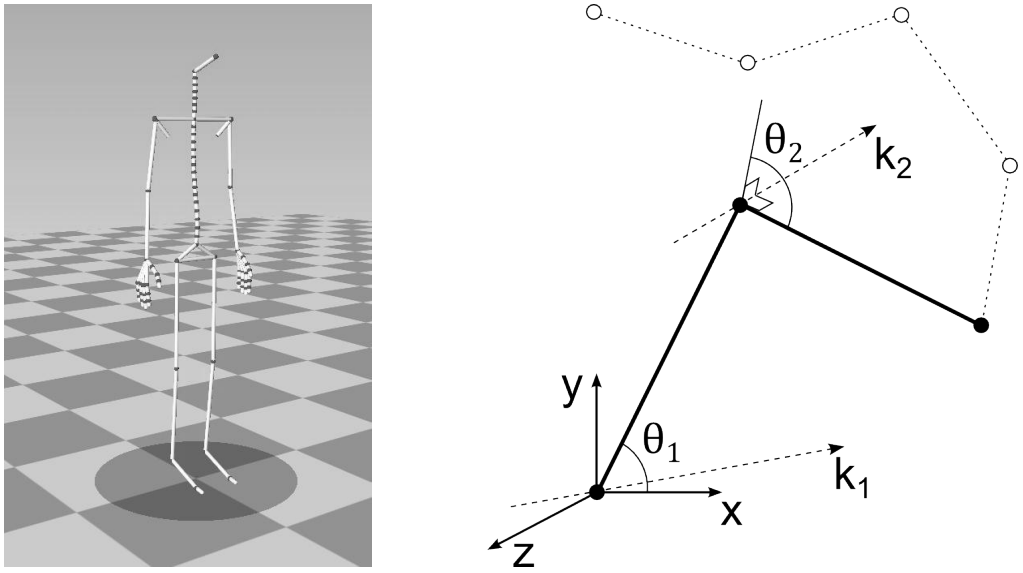


Figure 1.2.1: *A simplified sketch of the manikin.*

freely in space. A set of joint values positioning the manikin in this way will be referred to as a pose or a joint vector.

The circles connected by dashed lines that can be seen in the right sketch of figure 1.2.1 represent the target points that the outermost point of the chain is meant to follow. This tip of the linked chain is in the manikin's case most often the left and right hands, respectively. These points will throughout the report be referred to as Tool Center Points (TCPs), a name that is more commonly used to describe how the tools of industrial robots are positioned and rotated.

Defining a TCP target position as a single point that one of the joints should match, typically gives multiple solutions. This can be seen in figure 1.2.2 where a simple robotic arm with two arm joints can position its tool in the same position in two different ways. In the figure there is also a third wrist joint that - besides positioning the tool in the same way - also enables the robot to match the rotation of the tool. Solving this kind of problem, where joint values have to be found to match a target point, is referred to as inverse kinematics.

In order to uniquely specify how the manikin is positioned, the target point can be defined as a vector of joint values that uniquely defines both the manikins pose and the target position. These joint vectors - or manikin poses - will be the starting point of the optimisation in this thesis.

1.3 General Problem Formulation

The starting point is thus a sequence of pre-calculated joint vectors that together make up a more or less loosely sampled path for the manikin to follow. These vectors are found by solving the problem of inverse kinematics and their corresponding poses are also calculated to be collision free, well balanced and as ergonomically correct as possible. It is due to the fact that the problem of inverse kinematics gives multiple solutions that these extra conditions can be considered. The task is now to use these sampled values and calculate a movement in time that satisfies conditions of speed and acceleration. Extra points can be added between the original poses in order to produce a smoother and more finely sampled path. But it is necessary to keep in mind when adding these extra points that they may not deviate too much from the original path since this might break the conditions that assured the path to be collision free, well balanced and ergonomically correct.

Adding the objective that the time the movement takes should be as short as possible the problem becomes a minimisation problem. The variables of the problem are not only all time values, but also all the angles, angular velocities and angular accelerations at all points in time. Remembering the number of joints of the manikin from section 1.2, about 160 joints, it is easy to see that the problem consists of a lot of variables. The problem is to find the set of values for these variables that minimise the time and still satisfy all of the constraints.

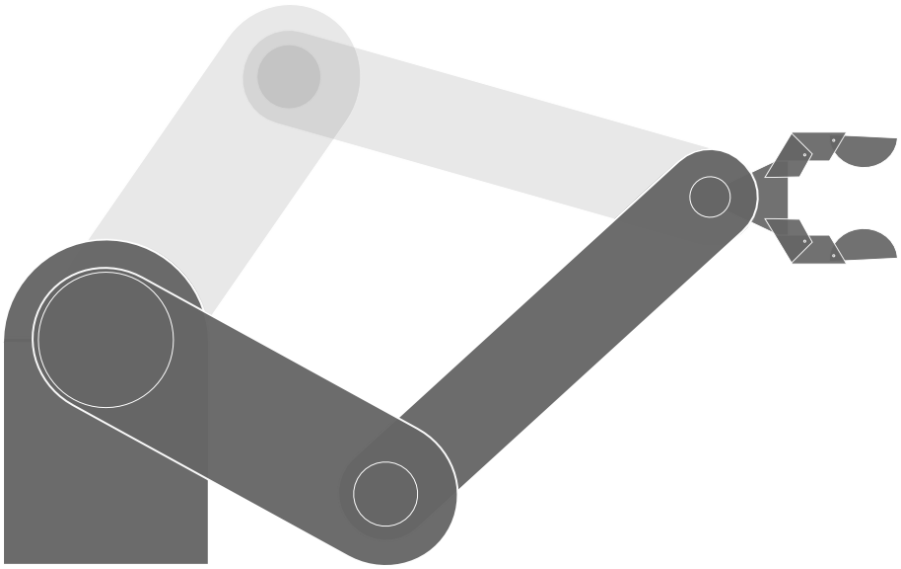


Figure 1.2.2: *The figure shows how a simple robotic arm with only two joints can have multiple valid joint value solutions for a single target point.*

1.4 Previous Work

The task of modelling human motion is a well-studied matter and a number of manikin projects have been developed. One example is the Jack human simulation system which was developed at the Center for Human Modeling and Simulation at the University of Pennsylvania beginning in the 1980s and later used and supported by NASA and the U.S. Army [2]. The simulation system has since then been bought and further developed by Siemens.

Other examples of off the shelf manikins are *RAMSIS* from Human Solutions [3] and *Human modeling* from Dassault Systèmes [4].

The problem with these manikins, and also simulations of robotic assembly, is that that it can require a lot of manual work to simulate a movement. The advantage with the IPS software developed at FCC is that feasible motions can be generated automatically for a number of different industrial robots working with rigid bodies. In order to include human simulation into this software the manikin project has been developed from the ground up. The IMMA project is a work in progress and a good deal of the work has been done by students who focus on different parts, adding new types of functionality to the project.

Previous work at FCC concerning the manikin project includes collision avoidance, ergonomically set conditions and human population sampling [5, 6].

1.5 Contributions

A big part of the previous work at FCC is used as a starting point for this thesis. After calculating collision free poses of the manikin that are well balanced and chosen to be as ergonomic as possible, the motion is shown using a fixed velocity that takes the manikin through the sequence of poses using linear interpolation of the joint values. The goal of this thesis is to improve this motion generation, producing more smooth and realistic motions. This step will be a post processing step where the previously calculated poses are used to ensure that the resulting movement still is feasible. But dividing the optimisation into two parts like this also generates some problems. If the final path does not exactly follow the initial sequence of poses, how is it possible to ensure that the final motion does not collide with fixed objects? This question will be addressed by adding constraints to the movement of the manikin.

The task of improving the motion generation by adding velocity and acceleration limits is meant to produce smooth and efficient paths. The found solution is important in it self since the path

together with its calculated time values can be animated and this smooth motion will look more realistic than a constant speed, linear animation. But including velocities and accelerations in the final solution is also an important first step towards further improving the ergonomic conditions. When trying to calculate ergonomic motions it is important to consider both the velocity and the acceleration of the movement. To take an example, a number of fixed poses, such as holding a screwdriver in different positions, might not be very straining. But if a series of these positions are connected to a motion which is repeated multiple times with high speed and large forces and moments, as when screwing something together, it might not be a good movement ergonomically.

1.6 Limitations

This thesis mainly concerns implementation of generalised theory. Since this theory will be used in connection to a single concrete case this will in some cases lead to loss of generality. That said, it has been a goal to keep the implementation as general as possible, in order to make it possible to add other conditions at a later stage.

1.7 Reading guidance

The next chapter will contain general theory needed to solve the minimisation problem. An alternative to reading the theory first is to directly skip to section 2.6 where the problem is formulated theoretically and returning then to the other theory sections - sections 2.1 through 2.5 - when needed.

In chapter 3 the method for solving the optimisation problem stated in section 2.6 will be presented and motivated. A number of different features added to the optimisation will also be presented. Solutions to a variety of problems will be presented in chapter 4 focusing mainly on simple problems that are chosen to highlight different characteristics of the found solution.

Finally, in chapter 5, conclusions will be drawn about how the motion generation performs and areas of future studies will also be presented and discussed.

2 Theory

In order to optimise the control of the manikin there are two main steps that have to be performed. One is of course the optimisation itself, which has been largely based on the IPOPT optimisation tool that will be presented in section 2.3. But in order to be able to optimise the problem, it has to be discretised first. The discretised approximation is produced by following the framework of the Galerkin method, outlined in section 2.1.

2.1 The Galerkin method

In the book Computational Differential Equations by K. Eriksson et al. [7], Galerkin's method for solving a general differential equation is said to be "based on seeking an (approximate) solution in a finite-dimensional space spanned by a set of basis functions which are easy to differentiate and integrate, together with an orthogonality condition determining the coefficients or coordinates in the given basis."

This means that in order to find an approximate solution to the differential equation it is necessary to first determine what set of basis functions this solution should belong to. This finite-dimensional space of basis functions is most often some set of discrete functions. The orthogonality condition means that the solution in some way is the best approximation given the type of functions used for the discretisation.

The Galerkin method can be used to discretise the following ordinary differential equation (ODE).

$$\dot{u}(t) + a(t)u(t) = f(t) \tag{2.1.1}$$

Here $a(t)$ and $f(t)$ are known functions of time and $u(t)$ is the sought unknown function. In order to find this unknown function the first step is to multiply equation (2.1.1) with a family of test functions $v(t)$ and then integrate these expressions over time.

$$\int_0^T \left(\dot{u}(t) + a(t)u(t)\right) v(t)dt = \int_0^T f(t)v(t)dt \tag{2.1.2}$$

This equation is identical to equation (2.1.1) if the equality holds for every test function $v(t)$. Instead of requiring that the equality should hold for all test function an approximation can be found by using a test space of functions. The test space is made up of all functions $v(t)$ that can be expressed as a sum over a set of the selected base functions $\gamma_i(t)$.

$$v(t) = \sum_{i=0}^{N} v_i \gamma_i(t) \tag{2.1.3}$$

These base functions are typically chosen to be functions that are very simple to integrate, such as piecewise linear or piecewise constant functions. If these basis functions are also chosen to be mostly zero with little or no overlap between them then the integration of the sum becomes particularly simple.

A common choice of basis functions used in the finite element method (FEM) are the so called tent functions. These functions, that are piecewise linear and continuous functions, can be seen in figure 2.1.1 and have the characteristics of basis functions that make the integrals easy to compute. They are chosen to be zero at all but one of the discretised points and have little or no overlap, only to their nearest neighbour. In the single point where the basis function has a value it is one, which makes it very easy to determine the factors in front of each basis function in equation (2.1.3). The closest approximation to a function $u(x)$ in this basis is simply the sum of basis functions that have the corresponding discretised function values as factors.

$$\tilde{u}(x) = \sum_i u(x_i)\gamma_i(x) \tag{2.1.4}$$

Here γ_i is the basis function that equals 1 at x_i and is 0 for all other $x_{j\neq i}$. The basis functions that are used to define the solution space do not necessarily have to be identical to the basis functions

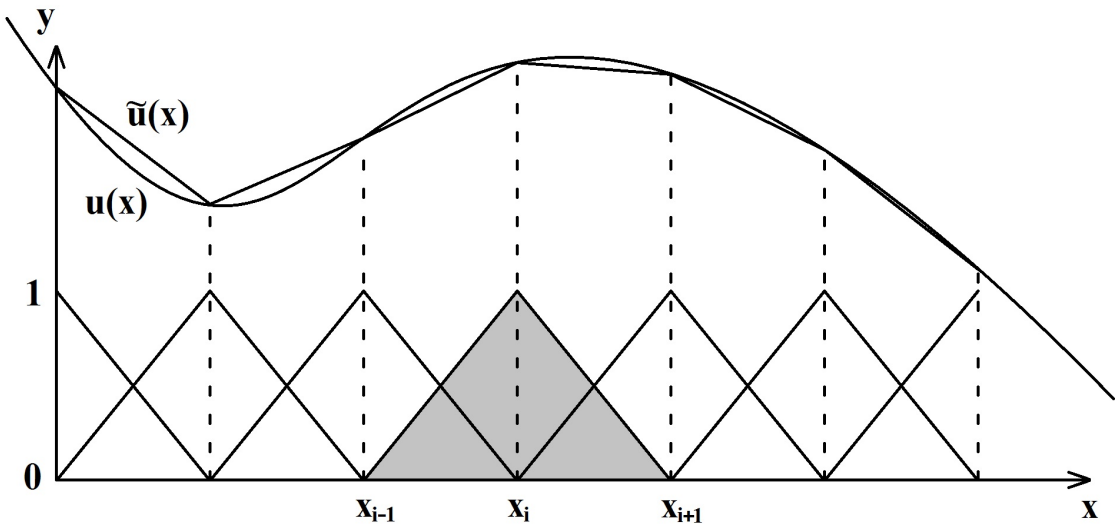


Figure 2.1.1: *The figure shows how piecewise constant basis functions can be used to get an approximation, $\tilde{u}(x)$, of a general function $u(x)$. The basis function marked with grey represents $\gamma_i(x)$.*

that define the test space. They are some times chosen to be identical for convenience, but in general, different Galerkin methods are obtained by choosing the two spaces in different ways.

2.2 General optimisation theory

In this section, where optimisation problems will be treated in a general way, problems with only inequality constraints are considered. This means that the problem formulation takes the following form:

$$\min_{x \in \mathbb{R}^n} \quad f(x) \quad \text{with} \quad c_i(x) \geq 0 \tag{2.2.1}$$

Only using inequality constraints will make the notation clearer and equality constraints can always be formulated using two inequality constraints.

$$c(x) = 0 \quad \Leftrightarrow \quad \begin{cases} c(x) \geq 0 \\ -c(x) \geq 0 \end{cases} \tag{2.2.2}$$

Optimisation problems of the kind defined in equation (2.2.1) can be divided into two main groups. If both the objective function $f(x)$ and the constraint functions $c(x)$ are linear, the problem is said to be a linear optimisation problem (also linear programming problem), and when the order of the functions are higher these problems are commonly referred to as nonlinear problems.

2.2.1 The dual problem and Lagrange multipliers

In constrained optimisation the problem as seen in equation (2.2.1), which is called the primal problem, can often be converted into a related problem referred to as the dual problem. When it comes to linear problems the primal and the dual version of the problem are closely linked. This linear case will be looked into more closely, since similar variables and terminology will also be used in the more general nonlinear optimisation.

A linear system of equations can be stated in the matrix form $A\mathbf{x} = \mathbf{b}$. Here A is the matrix of

coefficients, $\mathbf{x}$ is the vector of variables and $\mathbf{b}$ is the vector of scalar coefficients.

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}}_{\mathbf{b}} \tag{2.2.3}$$

In the same way a linear optimisation problem can be stated in matrix form in the following way.

$$\text{Minimise} \quad \mathbf{c}^T\mathbf{x} \quad \text{subject to} \quad A\mathbf{x} \geq \mathbf{b} \quad \text{with} \quad \mathbf{x} \geq 0 \tag{2.2.4}$$

This minimisation problem can be transformed into a related dual problem. This dual problem will use the vector of dual variables $\mathbf{y}$ and is defined as the following optimisation problem.

$$\text{Maximise} \quad \mathbf{b}^T\mathbf{y} \quad \text{subject to} \quad A^T\mathbf{y} \leq \mathbf{c} \quad \text{with} \quad \mathbf{y} \geq 0 \tag{2.2.5}$$

When solving the primal problem the dual problem can be used to find upper boundaries of the correct solution. This is because the following condition always holds for two vectors $\mathbf{x}$ and $\mathbf{y}$ of allowed values [8].

$$\mathbf{c}^T\mathbf{x} \geq \mathbf{b}^T\mathbf{y} \tag{2.2.6}$$

Vectors $\mathbf{x}, \mathbf{y}$ of allowed values are vectors that fulfill all constraints of their respective problems.

$$\begin{array}{llll} \mathbf{x} \in \mathbb{R}^n & \text{such that} & \mathbf{A}\mathbf{x} \geq \mathbf{b} & \text{and} \quad \mathbf{x} \geq 0 \\ \mathbf{y} \in \mathbb{R}^m & \text{such that} & \mathbf{A}^T\mathbf{y} \leq \mathbf{c} & \text{and} \quad \mathbf{y} \geq 0 \end{array} \tag{2.2.7}$$

Equation 2.2.6 holds for all alowed values but if two allowed vectors $\mathbf{x}$ and $\mathbf{y}$ are found that fulfil $\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{y}$ then these two vectors are optimal solutions to the primal and dual problems respectively.

In the same way as the dual variables $\mathbf{y}$ were introduced above, a vector of Lagrange multipliers λ can be introduced when solving a nonlinear optimisation problem.

For the nonlinear optimisation problem in equation (2.2.1) the Lagrangian function $L(x, \lambda)$ is defined as follows.

$$L(x, \lambda) = f(x) + \sum \lambda_i c_i(x) \tag{2.2.8}$$

Using this Lagrangian function, which contains a sum over all constraints c_i, the constrained optimisation problem is now an unconstrained optimisation problem where the constraints are included in the objective function. This function is used in the optimisation and the goal is to find a point where the Lagrangian multipliers fulfil the Karush-Kuhn-Tucker conditions. These KKT conditions will be presented in the next section.

2.2.2 The Karush-Kuhn-Tucker conditions

Using the notations of the problem formulation in equation (2.2.1) the following four equations must hold for a Lagrange multiplier vector λ^* in order for the point x^* to be a first order KKT (Karush-Kuhn-Tucker) point.

$$c(x^*) \geq 0 \tag{2.2.9a}$$

$$g(x^*) = J(x^*)^T\lambda^* \tag{2.2.9b}$$

$$\lambda^* \geq 0 \tag{2.2.9c}$$

$$c_i(x^*)\lambda_i^* = 0 \tag{2.2.9d}$$

Satisfying the four KKT conditions in equations (2.2.9) is one of the main conditions a point x^* has to fulfil in order to be locally optimal. If the problem is convex the KKT conditions are even sufficient conditions for optimality.

So what do the four equations of (2.2.9) mean? The first equation, (2.2.9a), shows that the point x^* is a feasible point where all inequality constraints are satisfied. The second condition,

equation (2.2.9b), is a stationarity condition. It assures alignment of the gradient of the objective function with the gradients of the active constraints. The third equation, (2.2.9c), makes sure that the aligned gradients are not antiparallel. This is true for non-negative multipliers, λ_i^*. The last equation, (2.2.9d), is the condition that connects all constraints with their Lagrange multiplier. Here it is stated that a constraint $c_i(x^*)$ either can be active, $c_i(x^*) = 0$, or else the corresponding multiplier, λ_i, has to be equal to zero. That means that inactive constraints do not affect the stationarity condition of equation (2.2.9b).

2.2.3 The simplex method

A special feature of linear problems is that their solution is always at a vertex of the constraints. That is a point where a subset of the constraint functions are equal to zero (these are referred to as active constraints) and the rest of the constraints are strictly positive (inactive constraints). In order to understand this characteristic the problem can be studied geometrically, only using two variables for the optimisation. The linear optimisation function is in this case a slanted plane and all constraints are represented with straight lines. If a polygon is drawn using any number of these constraints in order to define the allowed area, the maximum (and the minimum) of the objective function will be found at one of the corners of the polygon. The two lines that intersect at this maximal point represent the active constraints and all the other constraints are said to be inactive. It is the two active constraints that effectively set the upper boundary. Even if all the other inactive constraints were removed, the solution to the problem would still be exactly the same. This two dimensional case can be generalised to higher dimensions and can be used for all types of linear optimisation.

This feature of linear optimisation problems is exploited by the simplex method, which is a method that is often used to solve linear problems. When optimising the motion of the manikin, the resulting optimisation problem is not a linear problem so the simplex method is not a method that can be used in this thesis. The method is nonetheless included here in order to show the connections between linear and nonlinear optimisation, and to highlight the problems an optimisation method must overcome.

The simplex method tries to find the constraints that are active at the optimal solution and iteratively cycles through subsets of active constraints in order to find the correct constraints and the optimal solution. Since this is a combinatorial problem with respect to the number of constraints the execution time for the worst case scenario grows exponentially. That said, the algorithm works well in practice and is a powerful algorithm that is often used to solve linear problems.

By not directly tackling the combinatorial problem of finding the correct set of active constraints, the interior point methods of section 2.2.4 can get away from the worst case exponential rate of the execution time.

2.2.4 Interior point methods

Interior point methods are a set of optimisation methods that can solve optimisation problems stated in a very general form, both linear and nonlinear problems. The interior point method that will be used here is called Ipopt and will be presented in section 2.3. This section will instead present interior points methods in a general way, the problems they face and how they tackle them.

Interior point methods must find the subset of active constraints that define the solution in order to solve a linear optimisation problem, just as the simplex method of section 2.2.3 did. However these methods do not search from vertex to vertex, but instead - as the name implies - use a sequence of points in the *interior* of the allowed region. The points all follow a smooth path, iteratively approaching the boundary and the optimal solution. Instead of iteratively searching for the optimal solution by trying different combinations of active constraints, interior point methods select their subset of active constraints by finding values of the primal variables x_i and the dual variables λ_i that satisfy the KKT conditions of equation (2.2.9).

Interior point methods are also known under the name barrier methods. This is because the sequence of points that lead up to the final solution are the solutions of consecutive barrier problems. It is these barriers that keep the solution in the allowed area during the optimisation and they will be more clearly explained in the following chapter.

2.3 The basics of Ipopt

IPOPT (Interior Point OPTimizer) is a publically available optimising tool, designed to solve constrained optimisation problems [9].

It is based on solving a sequence of barrier problems, as mentioned in section 2.2.4. These barriers are functions that are designed to keep all investigated points within the allowed area. Typical characteristics of barrier function are that they mimic the objective function at points far away from the boundary in the interior of the allowed region, but then rise to infinity for points approaching the boundary. It is also desirable that the function is well behaved. In this case a well behaved function is a function that is smooth, which means that it is differentiable, and preferably that it is even easy to differentiate. A few different barrier functions have been suggested, but the most commonly used function is the logarithmic barrier function.

$$B(x, \mu) = f(x) - \mu \sum_{i=1}^{m} \ln(c_i(x)) \tag{2.3.1}$$

Here the sum is taken over m which is the total number of constraints c_i. If the barrier parameter, μ, is chosen to a small value, this function mimics the original objective function $f(x)$, except close to the boundaries of the feasible region where the function rapidly grows towards infinity. A bigger value of the barrier parameter μ makes the barrier function smoother and in general this means that the function is easier to optimise.

The chosen barrier function of equation (2.3.1) is not only differentiable but also very easy to differentiate and its gradient is

$$\nabla B(x, \mu) = \nabla f(x) - \mu \sum_{i=1}^{m} \frac{1}{c_i(x)} \nabla c_i(x) \tag{2.3.2}$$

The idea of using this barrier function is that it keeps the points in the interior of the allowed region and at the same time it is possible to gradually approach the optimal solution. If the function is sequentially minimised as the barrier parameter is lowered, the solutions of these unconstrained problems should converge to the original constrained optimisation of $f(x)$.

2.4 Ipopt input

In order to be able to interface with IPOPT it is important to know how IPOPT handles optimisation problems. There is an extensive introduction to IPOPT that among other things describes the interface and user options. IPOPT can be used to solve general nonlinear programming problems of the following form [10].

$$\min_{x \in \mathbb{R}^n} \quad f(x) \tag{2.4.1a}$$

$$\text{such that} \quad g^L \leq g(x) \leq g^U \tag{2.4.1b}$$

$$x^L \leq x \leq x^U \tag{2.4.1c}$$

Compared to the general problem statement in equation (2.2.1) this formulation is stated with both upper and lower bounds on all constraints and variables. But this formulation is just as general, for unbounded variables the lower or upper bounds are set to negative or positive infinity.

IPOPT requires some third party components, including BLAS (Basic Linear Algebra Subroutines), LAPACK (Linear Algebra PACKage) and a sparse symmetric indefinite linear solver. All of these components are available as open source versions and a number of different alternatives are listed in the IPOPT introduction [10].

In order to optimise a given problem, IPOPT uses not only the objective and constraints themselves but also needs the first and second derivatives of the functions. This means that for a given problem it is also necessary to calculate the gradient of the objective $\nabla f(x)$, the Jacobian of the constraints $\nabla g(x)^T$ and the Hessian of the Lagrangian function which includes both $\nabla^2 f(x)$ and $\nabla^2 g_i(x)$.

2.5 Forward and inverse kinematics

Forward kinematics refers to the problem of finding the positions in space of all the rigid bodies that make up the model, given all joint values. The tree-structure of the model makes it possible to solve this problem iteratively. Starting at the root, the position of the following link can be found by rotating the coordinates using the current joint value.

A general rotation of a vector $\mathbf{u}$ about some unit vector $\mathbf{k}$ by an angle θ can be described by Rodrigues' rotation formula.

$$\tilde{\mathbf{u}} = \mathbf{u}\cos\theta + (\mathbf{k}\times\mathbf{u})\sin\theta + \mathbf{k}(\mathbf{k}\cdot\mathbf{u})(1-\cos\theta) \tag{2.5.1}$$

It is often convenient to express this rotation using a rotation matrix $\mathbf{R}$ so that the equation can be written $\tilde{\mathbf{u}} = \mathbf{R}\mathbf{u}$. Using the versine function $\operatorname{vers}\theta = (1-\cos\theta)$ to get a more compact expression, this rotation matrix will have the following form.

$$\mathbf{R}(\mathbf{k},\theta) = \begin{bmatrix} k_x^2\operatorname{vers}\theta + \cos\theta & k_yk_x\operatorname{vers}\theta - k_z\sin\theta & k_zk_x\operatorname{vers}\theta + k_y\sin\theta \\ k_xk_y\operatorname{vers}\theta + k_z\sin\theta & k_y^2\operatorname{vers}\theta + \cos\theta & k_zk_y\operatorname{vers}\theta - k_x\sin\theta \\ k_xk_z\operatorname{vers}\theta - k_y\sin\theta & k_yk_z\operatorname{vers}\theta + k_x\sin\theta & k_z^2\operatorname{vers}\theta + \cos\theta \end{bmatrix} \tag{2.5.2}$$

In order to find the rotations of all rigid bodies in the model, the angular values in each joint have to be used to propagate the rotations forward through the tree structure. The rotational matrix that connects a parent i and its child $i+1$ will be denoted ${}_i\mathbf{R}_{i+1}$ and the rotation matrix that defines the global orientation for body i is denoted $\mathbf{R}_i$. These matrices can be multiplied to give the iterative formula for calculating the global orientations of all the rigid bodies in the model.

$$\mathbf{R}_{i+1} = \mathbf{R}_i\,{}_i\mathbf{R}_{i+1}(\mathbf{k}_{i+1},\theta_{i+1}) \tag{2.5.3}$$

In this formula $\mathbf{R}_i$ denotes the global orientation of the parent and $\mathbf{R}_{i+1}$ the global orientation of the child. The matrix ${}_i\mathbf{R}_{i+1}(\mathbf{k}_{i+1},\theta_{i+1})$ is the rotational matrix between the two global orientations and is dependent on the axis of rotation $\mathbf{k}_{i+1}$, which is pre-definded, and on the joint angle θ_{i+1}. Since the vector $\mathbf{k}_{i+1}$ is both pre-defined and constant the notation can be cleared up by including it in the rotational matrix, ${}_i\mathbf{R}_{i+1}(\theta_{i+1})$. This matrix can also be split up into a constant matrix containing the rotation that brings the coordinate system to the neutral position $\theta_{i+1}=0$ and a second part that describes the rotation as a function of θ_{i+1}.

$${}_i\mathbf{R}_{i+1}(\theta_{i+1}) = {}_i\mathbf{R}_{i+1}(0)\mathbf{S}_{i+1}(\theta_{i+1}) \tag{2.5.4}$$

Note that the rotational matrix $\mathbf{S}_{i+1}(\theta_{i+1})$ is a pure rotation of the same form as $\mathbf{R}$ in equation (2.5.2). The matrix depends on the rotational vector and this dependency could be explicitly written $\mathbf{S}_{i+1}(\theta_{i+1}) = \mathbf{S}(\mathbf{k}_{i+1},\theta_{i+1})$ in order to more clearly see the resemblance with $\mathbf{R}(\mathbf{k},\theta)$.

The calculations of the global positions have a similar iterative form and use the global rotational matrices together with local vectors $\mathbf{b}_i$. The local vector $\mathbf{b}_i$ is a vector between the child frame and the parent frame, and is given in the coordinates of frame i. The two equations of global positions and rotations are often used together and are stated below.

$$\begin{cases} \mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{R}_i\mathbf{b}_i \\ \mathbf{R}_{i+1} = \mathbf{R}_i\mathbf{S}(\mathbf{k}_{i+1},\theta_{i+1}) \end{cases} \tag{2.5.5}$$

The notations for the global positions, $\mathbf{x}_i$, and the global rotations, $\mathbf{R}_i$, are the same as in figure 2.5.1 and the constant matrix ${}_i\mathbf{R}_{i+1}(0)$ has been dropped for clarity. The index i represents the parent and index $i+1$ the child, which in the figure is shown by index i representing the rigid body closer to the base. The figure also shows local coordinate systems $\mathbf{F}_i$ and $\mathbf{F}_{i+1}$. In global coordinates these local coordinate systems $\mathbf{F}$ are rotated just as $\mathbf{R}$ and translated with $\mathbf{x}$. These are used to express local vectors such as $\mathbf{b}_i$ which is a local vector in $\mathbf{F}_i$.

For forward kinematics the joint values θ_i are set to a certain pose and the procedure to find the global values is a strictly iterative procedure using the two equations of (2.5.5). The inverse problem uses the same set of equations and tries to find the joint values that match up to a given position. This is a more complicated problem and is typically solved using some kind of iterative method that uses the error for intermediate steps to converge to one of the multiple correct solutions.

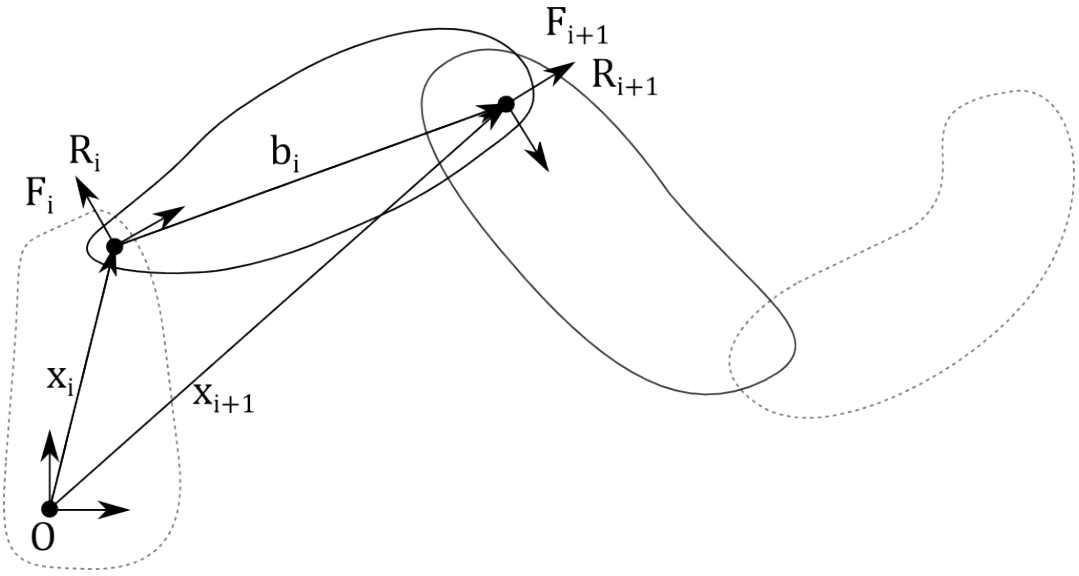


Figure 2.5.1: *The sketch shows what notations are used for rotations and positions, both globally and locally.*

2.6 Theoretical problem formulation

Remembering the general problem formulation of section 1.3 the problem will in this following section be treated in a more theoretical manner.

Given a number of target positions the problem is to determine what control function, or in this case what acceleration, will produce an optimal time stamped path through the target positions. For now the term optimal path only describes some broad sense of optimality, but this will be brought up and more clearly applied in sections 2.6.3 and 2.6.4.

2.6.1 A general Optimal Control formulation

The problem can be stated as an optimal control problem and in general this kind of problem can be formulated in the following way:

$$\min J(y,u,t) \quad \text{such that} \tag{2.6.1a}$$

$$\frac{dy}{dt} = f(y,u,t) \tag{2.6.1b}$$

$$g(y,u,t) \geq 0 \tag{2.6.1c}$$

$$h(y(0),y(T)) = 0 \tag{2.6.1d}$$

where $J(y,u,t)$ is the cost functional to be minimised. The variable y denotes the position and equation (2.6.1b) determines how this variable varies in time. This means that in general when the goal is to solve an optimal control problem, the task is to find values of the position y, the control signal u and the time t in order to minimise the cost functional. This has to be done while fulfilling the differential equation (2.6.1b) and not breaking the inequality constraints g or the equality constraints h. For the equality constraints the notation of equation (2.6.1d) explicitly shows that the equality conditions can depend on both the initial and final values of the position.

2.6.2 The dynamics of the problem

Considering the general optimal control formulation of section 2.6.1 it is now possible to identify the different parts of the expression and determine what they look like in a concrete case.

Denoting the position x, the velocity v and the acceleration u, the dynamics of the manikin

problem can be stated in the following way.

$$\ddot{x}(t) = u(t) \tag{2.6.2a}$$
$$v_L \leq v(t) \leq v_U \tag{2.6.2b}$$
$$u_L \leq u(t) \leq u_U \tag{2.6.2c}$$
$$x(t_j) = x_j^{knot} \tag{2.6.2d}$$

Here the subscript L and U denote the lower and upper bounds, respectively. The variables x, v and u are all vector valued with one value for each of the m number of joints.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \in \mathbb{R}^m, \quad v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix} \in \mathbb{R}^m, \quad u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix} \in \mathbb{R}^m \tag{2.6.3}$$

Starting with the second order differential equation (2.6.2a), this can be rewritten into two first degree differential equations using the velocity v.

$$\frac{dx}{dt} = v \tag{2.6.4a}$$
$$\frac{dv}{dt} = u \tag{2.6.4b}$$

Combining the position x and the velocity v into a single variable $y = [x, v]^T$ makes it possible to also combine these two equations.

$$\frac{d}{dt}\begin{bmatrix} x \\ v \end{bmatrix} = \begin{bmatrix} v \\ u \end{bmatrix} \tag{2.6.5}$$

This equation describing the dynamic constraints can now be recognised as the differential equation in the optimal control formulation, equation (2.6.1b).

In the optimal control formulation the constraining of the problem is determined by equations (2.6.1c) and (2.6.1d), that represent constraints on the variables themselves. In the case of the manikin, the equality constraints, h, of equation (2.6.1d) set the positions, x, at an unknown time t_j to match a predefined fixed position x_j^{knot}.

$$h_j(y, u, t) = x(t_j) - x_j^{knot} = 0 \qquad \forall j \in \{1, \ldots, n\} \tag{2.6.6}$$

This type of constraint will often be used for the first and last points since these positions are most often well-defined points that the final solution should include. But these constraints can also be used for intermediate points that must be part of the solution. The subscript j is the index that denotes each of the n number of fixed positions.

As for the inequality constraints, g, these are used to set lower and an upper bounds on the velocity v and acceleration u. The bounds are set as an inequality constraint for each bound.

$$v(t) - v_L = g(y, u, t) \geq 0 \tag{2.6.7a}$$
$$-v(t) + v_U = g(y, u, t) \geq 0 \tag{2.6.7b}$$
$$u(t) - u_L = g(y, u, t) \geq 0 \tag{2.6.7c}$$
$$-u(t) + u_U = g(y, u, t) \geq 0 \tag{2.6.7d}$$

Inequality constraints are also used to ensure that all time differences, Δt_j, are positive.

$$\Delta t_j = t_j - t_{j-1} = g_j(y, u, t) \geq 0 \qquad \forall j \in \{1, \ldots, n\} \tag{2.6.8}$$

Returning to the dynamic constraints of equation (2.6.5) this can be written as a single matrix equation in a slightly different way.

$$\frac{d}{dt}\begin{bmatrix} x \\ v \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}}_{A}\begin{bmatrix} x \\ v \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{b} u \tag{2.6.9}$$

The matrix equation has now been written in a way that is easy to handle using the Galerkin method. The form of the equation can be recognised from equation (2.1.1) in the Galerkin theory section.

$$\dot{y}(t) + a(t)y(t) = f(t) \tag{2.6.10}$$

2.6.3 First problem formulation: minimise time

In the case of solving the manikin problem, the most simple and direct way to handle the cost functional is by setting

$$J(x, u, t) = T \tag{2.6.11}$$

which means that the goal is to have a motion that is as quick as possible since it is simply the final time, T, that is supposed to be minimised. This final time can also be written in terms of the variables t_j or Δt_j.

$$T = t_n = \sum_{j=1}^{n} \Delta t_j \tag{2.6.12}$$

2.6.4 Second problem formulation: minimise energy

One problem with only looking at a time optimal path is that the final solution might not be unique. This typically happens when the constraints mainly affect a subset of the joints. The system does not have to be complicated in order for this to happen.

As an example consider an arm with two identical joints. If the first joint has to move a longer distance than the second one, and the minimisation is only with respect to time, the path of the second joint is not uniquely defined. It has to move a certain (relatively short) distance in a certain (relatively long) time, and can do so in a number of ways. It could for example wobble back and forth as long as it arrived at the final position in time.

In order to avoid this behaviour an additional term is added to the cost functional. Instead of only minimising with respect to time, an energy term is also included. Here the energy is said to correspond to the integral of the squared control signal.

$$J(x, u, t) = T + c \int_0^T u(t)^T u(t) dt \tag{2.6.13}$$

The constant c can be chosen arbitrarily and it is a measure of the relative importance between the two objectives; to minimise time and to minimise energy. If this constant is set to a very small value, the time minimisation is unaffected and this additional energy criteria selects the minimum energy solution from all valid minimum time solutions. But the constant c can also be set to higher values to increase the importance of minimising the energy. This smoothes the control signal, bringing it away from the time optimal bang-off-bang solution which instantly changes between the extreme values. This effect will be seen and further discussed in section 4.3.

2.6.5 Adding spatial constraints

The constraints that have been described in section 2.6.3 that set lower and upper boundaries on the variables are easy to handle during the optimisation; it is a simple task to check if a variable is within a valid range. More complicated constraints might however be needed. Examples of this are constraints that limit the manikin's position in space, as opposed to limiting its joint values. The constraints that have been used up to now all deal directly with the variable vector, but if constraints are to be used to limit the manikin in space the positions of all the joints have to be calculated. This leads to constraints that are dependent on a number of different variables which in turn makes it harder to calculate the partial derivatives that are needed during the optimisation. In this section the expressions that are used in the optimisation using these spatial constraints will be presented.

All angles will be denoted q and spatial positions will be denoted by the point p, with coordinates p_x, p_y and p_z. Only rotational joints will be described since the manikin only contains a few

translational joints. The translational joints are also easier to handle since they linearly affect the position of other joints.

Spatial constraints will generally be given in the form of *ball-constraints* that set a maximum distance r the point p is allowed to deviate from a given fixed point p_0.

$$g(q) = \|p(q) - p_0\|^2 - r^2 = (p_x - p_{x_0})^2 + (p_y - p_{y_0})^2 + (p_z - p_{z_0})^2 - r^2 \tag{2.6.14}$$

The reason for stating the conditions in this way can be motivated by looking at how the original poses for the manikin are calculated. One of the conditions that have to be considered when calculating the poses is collision avoidance. These conditions are fulfilled by assuring that each rigid body has a sufficient amount of clearance with respect to its closest object. Using a distribution of points p that all are associated with a ball-constraint gives an approximation of the clearance.

The first and second partial derivatives the ball-constraint in equation (2.6.14) are given by the following expressions.

$$\frac{\partial g}{\partial q_j}(q) = 2(p_x - p_{x_0})\frac{\partial p_x}{\partial q_j} + 2(p_y - p_{y_0})\frac{\partial p_y}{\partial q_j} + 2(p_z - p_{z_0})\frac{\partial p_z}{\partial q_j} \tag{2.6.15a}$$

$$\frac{\partial^2 g}{\partial q_j \partial q_k}(q) = 2(p_x - p_{x_0})\frac{\partial^2 p_x}{\partial q_j \partial q_k} + 2(p_y - p_{y_0})\frac{\partial^2 p_y}{\partial q_j \partial q_k} + 2(p_z - p_{z_0})\frac{\partial^2 p_z}{\partial q_j \partial q_k} \tag{2.6.15b}$$

The partial derivatives of p with respect to the angular values q_j must also be calculated. Let the point p be given in the local coordinates of joint i, then the global coordinates of p can be written as

$$p = \mathbf{x}_i + \mathbf{R}_i p_b \tag{2.6.16}$$

where p_b is the local vector defining the point. This is the same expression that was found in equation (2.5.5). Taking the partial derivatives with respect to the angular values gives us the following expressions.

$$\frac{\partial p}{\partial q_j} = \frac{\partial x_i}{\partial q_j} + \frac{\partial R_i}{\partial q_j} p_b \tag{2.6.17a}$$

$$\frac{\partial^2 p}{\partial q_j \partial q_k} = \frac{\partial^2 x_i}{\partial q_j \partial q_k} + \frac{\partial^2 R_i}{\partial q_j \partial q_k} p_b \tag{2.6.17b}$$

Taking the partial derivatives of the recursive equation (2.5.5) makes it possible to construct a recursive algorithm to calculate all the partial derivatives of x and R above.

2.6.6 Summary of the problem formulation

This section summarises the theory chapter by quickly going through the formulation of the problem in its current form.

The objective function has now taken the following form:

$$\min J(y, u, t) = \min \left(T + c \int_0^T u(t)^T u(t) \right) \tag{2.6.18}$$

Multiplying equation (2.6.5) with some pair of test functions $r(t) = [r_x(t), r_v(t)]^T$ and integrating the expression, the kinetic equations can be written in integral form.

$$\int_0^T \frac{d}{dt} \begin{bmatrix} x \\ v \end{bmatrix} \cdot r(t) dt = \int_0^T \left(\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \right) \cdot r(t) dt \tag{2.6.19}$$

A linear combination of basis functions discretised at times τ_i can be used to describe both the test functions and the functions representing the position, velocity and acceleration. The basis functions

used to define the solution and test spaces are denoted $\phi_i(t)$ and $\gamma_i(t)$, respectively.

$$x(t) = \sum_{i=0}^{N} x(\tau_i)\phi_i(t) \tag{2.6.20a}$$

$$v(t) = \sum_{i=0}^{N} v(\tau_i)\phi_i(t) \tag{2.6.20b}$$

$$u(t) = \sum_{i=0}^{N} u(\tau_i)\phi_i(t) \tag{2.6.20c}$$

$$r(t) = \sum_{i=0}^{N} r(\tau_i)\gamma_i(t) \tag{2.6.20d}$$

Inserting the linear combinations into equation (2.6.19) this gives the differential equations as a sum of integrals.

$$\sum_{i=1}^{N} \int_{\tau_{i-1}}^{\tau_i} \frac{d}{dt} \begin{bmatrix} x \\ v \end{bmatrix} \cdot r(t)dt = \sum_{i=1}^{N} \int_{\tau_{i-1}}^{\tau_i} \left(A \begin{bmatrix} x \\ v \end{bmatrix} + bu \right) \cdot r(t)dt \tag{2.6.21}$$

Section 3.4 describes how the set of discretisation points $\{\tau_i\}$ are chosen. It should be noted that they are added as extra points so that the set of discretisation points $\{\tau_i\}$ always includes the set of fixed points $\{t_j\}$.

The constraints on the variables have to be fulfilled at these discrete points, so using the same notation as above the constraints can be written as follows.

$$v_L \leq v(\tau_i) \leq v_U \quad \forall i \in \{1, \ldots, N\} \tag{2.6.22a}$$

$$u_L \leq u(\tau_i) \leq u_U \quad \forall i \in \{1, \ldots, N\} \tag{2.6.22b}$$

$$x(t_j) = x_j^{knot} \quad \forall j \in \{1, \ldots, n\} \tag{2.6.22c}$$

Here N is the number of discretisation points and n is the number of fixed points.

3 Method and Implementation

This chapter will cover how the problem has been implemented and present the different types of functionality that have been added.

3.1 Implementation

The optimisation tool of the project is generally a stand-alone project which makes use of the third party IPOPT code. The project as a whole has however been closely linked to the IMMA project at FCC and makes use of the functionality and visualisation environment of this bigger project.

The project has been implemented using the C++ programming language just as the IMMA project, even if many of the simple examples shown in the result chapter have been generated using an early version implemented in MATLAB. For these simple illustrative examples it has been convenient to make use of the MATLAB plotting tools.

3.2 Choosing method

As mentioned in the introduction, simulations of industrial robots is a big research area at FCC and some of the work is quite close to the matters discussed in this thesis. More specifically there has been work done on controlling the motion of an industrial robot [11]. In this section the method used in the robot controller will be studied briefly and this comparison will hopefully give some motivation to the choice of method.

The problems that are faced in the robot control case are generally the same ones as in the manikin's case. The goal is to pass through a series of points as quickly as possible. This problem is in the robot's case solved by trying to find the exact optimal solution. This is done by taking the most simple problem - where the series of points only contains two points - and dividing this subproblem into different cases; finding the optimal solution for each one of them. These subproblems can then be used to build up the final solution for the whole series of points.

Only considering two points means that the problem is to as quickly as possible get from point A, starting with an initial velocity v_A, to point B, with a terminal velocity v_B. Which of the cases the solution to this subproblem falls into depends on the distance between the points and the target velocities. The optimal solution will however always be a bang-off-bang solution, only alternating between maximal, minimal or zero acceleration. Merging the solutions to the subproblems into a single solution has to be done iteratively. This is because the solutions to the subproblems might affect the neighbouring intervals if the distance between point A and B is so short that the terminal velocity v_B can not be reached.

This procedure which is used to find the optimal solution is quite powerful and is a good way to find the time optimal solution. However in the manikin's case it is important to be able to add other types of measures for the optimality - a human motion is not always time optimal. The robot controller is finely tuned for time optimisation but would have problems adjusting to other types of optimality conditions. It is also desireable to be able to add other types of variable constraints. For the simple example above, with two fixed points A and B and a motion constrained by constant velocity and acceleration limits, it is relatively easy to find the analytical solution to the subproblem. If some of these constraints were relaxed it might be much more difficult, if not impossible, to find the analytical solution to the subproblem and to combine solutions from multiple subproblems.

There is a need in the case of the manikin for a very general solver that easily can handle not only different optimality conditions but also different kinds of variable constraints. This is why the problem is treated as a set of ordinary differential equations that are solved by seeking numerical approximations to the optimal functions.

3.3 Discretising the problem

The discretisation of the problem uses the Galerkin approximations that are described in section 2.1 and summarised in section 2.6.6. Using piecewise constant acceleration and piecewise linear and continuous positions and velocities the expressions used to calculate the velocity v_i and the acceleration a_i take the following form.

$$x_i - x_{i-1} = \Delta t_i \frac{v_i + v_{i-1}}{2} \tag{3.3.1a}$$

$$v_i - v_{i-1} = \Delta t_i a_i \tag{3.3.1b}$$

This discretisation is a mixture of the Galerkin version dG(0) for the acceleration and cG(1) for the velocity and position. The notation dG(0) stands for discontinuous Galerkin of the zeroth order, which means that it is a function that instantly changes between constant values. The cG(1) is instead a continuous function of first order functions, which means that it is a continuous piecewise linear function.

In section 4.1 the kinetic equations of equation (3.3.1) are compared with a more symmetric way of calculating the velocity and acceleration where they are not a mixture of Galerkin versions. Using simple examples it is shown how this difference affects the final solution.

3.4 Adding points to the path

The kinematic equations (3.3.1) that were used in section 3.3 in order to update the velocity and acceleration are straight forward first derivative approximations of their respective functions. These approximations work very well when the movement is not very complicated, but when the acceleration changes abruptly or very often it can be important to see how the variables change between the fixed points. In order to obtain more accurate approximations of all the variables that are associated with the movement, one solution could be to increase the order of the Galerkin functions. Another solution, which is the one that will be used here, is to use the same approximations but instead insert additional intermediate points. These new points obey the same velocity and acceleration conditions as the original ones; the difference is that they are not fixed to predetermined positions. These extra points do not have time values of their own but are held in place by evenly dividing up the affected time segment. It is basically the same equations as in equation (3.3.1), the only difference is that the extra time values are not used as variables.

$$x_{i,j} - x_{i,j-1} = \frac{1}{m_i} \Delta t_i \frac{v_{i,j} + v_{i,j-1}}{2} \tag{3.4.1a}$$

$$v_{i,j} - v_{i,j-1} = \frac{1}{m_i} \Delta t_i a_{i,j} \tag{3.4.1b}$$

Here each section i is divided into m_i parts and the notation (i, j) denotes section i and subsection j. In the report subsections will also be denoted using a single index, but since they all are ordered there is no ambiguity when using the two different notations. In section 4.2 it is shown how these extra points affect the final solution.

3.5 Adding an energy term

In section 2.6.4 the theory behind adding an extra energy term to the objective function was presented. In the discrete case the integration becomes a sum which means that the objective function will take the following form.

$$J = \sum_{i=1}^{n} \Delta t_i + c \sum_{i=1}^{n} u_i^2 \tag{3.5.1}$$

The primary goal is to always obtain a unique solution to the optimisation problem - if multiple control signals can solve the problem the one which uses the least amount of energy is selected. In section 4.3 a simple example that exhibits this kind of behaviour will be presented. This example will also show how different values of the constant c can change the shape of the control signal.

4 Results and Discussion

The main part of this chapter will show a number of solutions to engineered problems in order to highlight different characteristics of the found solutions. To easily show what the found solution of the optimisation problem looks like, the model will initially be very simple, made up of only a single or very few joints. Using a single joint makes it possible to use graphs to show the how the position, velocity and acceleration develops over time. There will also be graphs showing more complicated and interesting multiple joint movements, but even two joints affect the clarity of the graphs.

4.1 Discretisation artefacts

The first thing that was brought up in the theory chapter was the Galerkin method of section 2.1 and the issue of how the problem should be discretised. In some cases a small difference in how the problem is discretised can show up as a big difference in the final solution. The discretisation and the kinematic expressions that are used to calculate the velocity and acceleration are stated in section 3.3. In this section this choice is motivated by first looking at a more basic form of the kinetic expressions and the artefacts that these equations bring with them.

In figure 4.1.1 the solution of a simple problem is shown. The problem consists of a single variable which can be thought of as a position of a rigid body or the angle of a single joint. The objective is to move the object from the starting position zero, where it is at rest, to position nine via the intermediate integer values. This should be done as fast as possible while still fulfilling the velocity and acceleration constraints. These constraints are in this case set to be $-2 \leq v \leq 2$ and $-1 \leq a \leq 1$. These values are quite arbitrary and so are the units, which are set to be in meters and seconds or alternatively radians and seconds.

For this simple example it is easy to foresee what the optimal solution should look like. The object should simply accelerate as fast as possible and if it reaches its maximum velocity it should hold this speed until it reaches its final destination. There are no constraints on the final velocity so there is no need for a deceleration phase. Looking at the figure all the correct characteristics described above are fulfilled, but it is also clear that there is something wrong with the acceleration. When the velocity is held constant the acceleration should be zero, but in the figure the acceleration starts to oscillate - alternating between the maximum and the minimum of the allowed acceleration limits.

This behaviour has to do with how the problem is discretised. In this case the discrete expressions that are used to calculate the velocity and acceleration have the following form.

$$x_i - x_{i-1} = (t_i - t_{i-1})\frac{v_i + v_{i-1}}{2} \tag{4.1.1a}$$

$$v_i - v_{i-1} = (t_i - t_{i-1})\frac{a_i + a_{i-1}}{2} \tag{4.1.1b}$$

This is a simple approximation of the first order derivatives of the form $v = \Delta x/\Delta t$ and $a = \Delta v/\Delta t$ together with an approximation of the central value using the mean of two neighbouring ones. The desired behaviour, that zero acceleration should give constant acceleration, is fulfilled but there is also another case that gives constant velocity. Since the equations only contain the mean acceleration value of two consecutive points in time, the velocity stays constant if $a_{i-1} = -a_i$. Remembering the oscillation from figure 4.1.1 it can also be noted that the acceleration can not drop down to zero instantly in order to keep the velocity constant. If the previous value is maximal the acceleration must take the minimal value in order to bring the sum of the two values down to zero. The oscillating pattern will emerge with alternating positive and negative values that all try to compensate for the previous acceleration value.

This problem is fixed by directly using the intermediate acceleration values as variables instead of the mean values used in equation (4.1.1b). The same notation a_i is used for these variables, the only change is that the number of acceleration variables is decreased by one. Removing redundant variables can also be done for the time variables. Instead of clamping the first time value to zero and calculating the time difference between two positions by subtraction, the time differences Δt_i are used as variables just as described in equation (2.6.8). This reduces the number of time

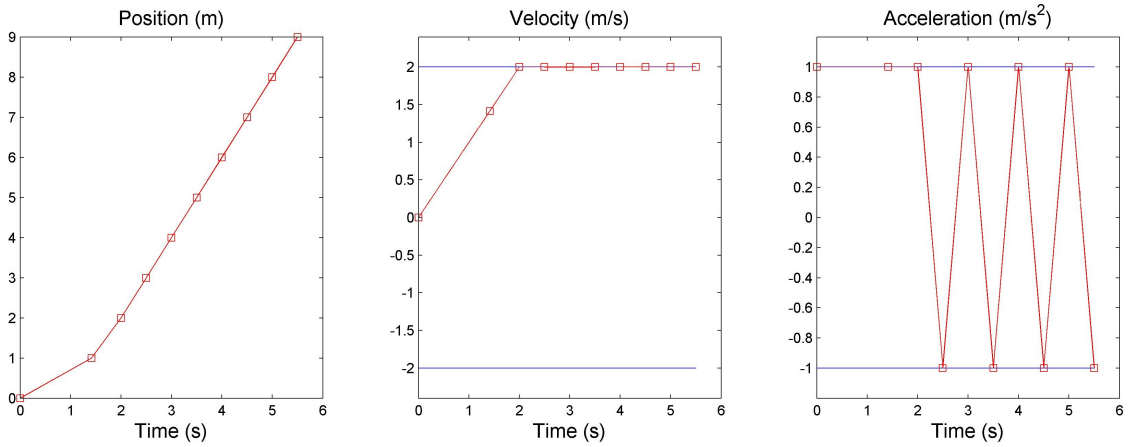


Figure 4.1.1: *The three graphs show the position, velocity and acceleration of a single joint that is set to move through the fixed points marked by squares. These fixed positions are all integer values from 0 to 9. The figure shows how the solution to the problem can oscillate without changing the velocity when the problem is discretised using equations* (4.1.1).

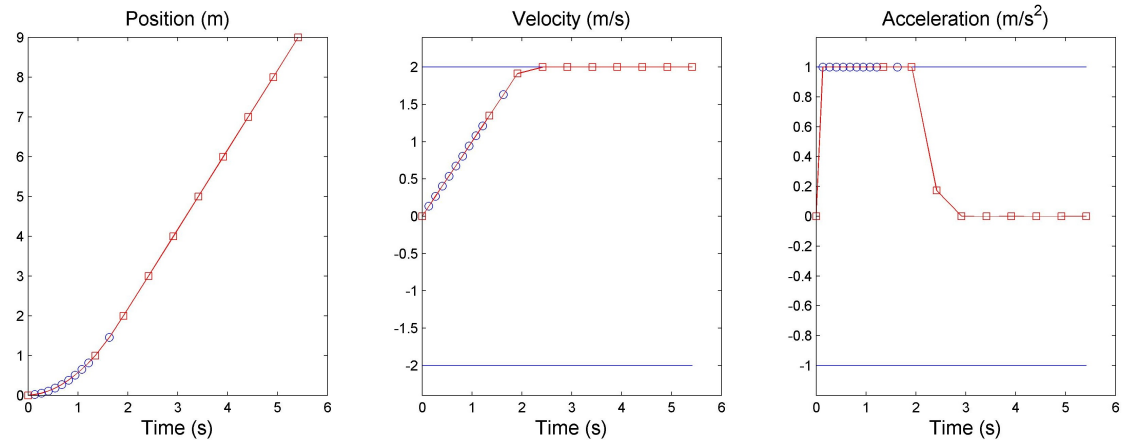


Figure 4.1.2: *The three graphs show the position, velocity and acceleration of a single joint that is set to move through the fixed points marked by squares. The discretisation is here defined by equations* (3.3.1) *and this removes the oscillations seen in figure 4.1.1. The circles mark extra points that are added to more clearly show how the motion changes at the start of the motion.*

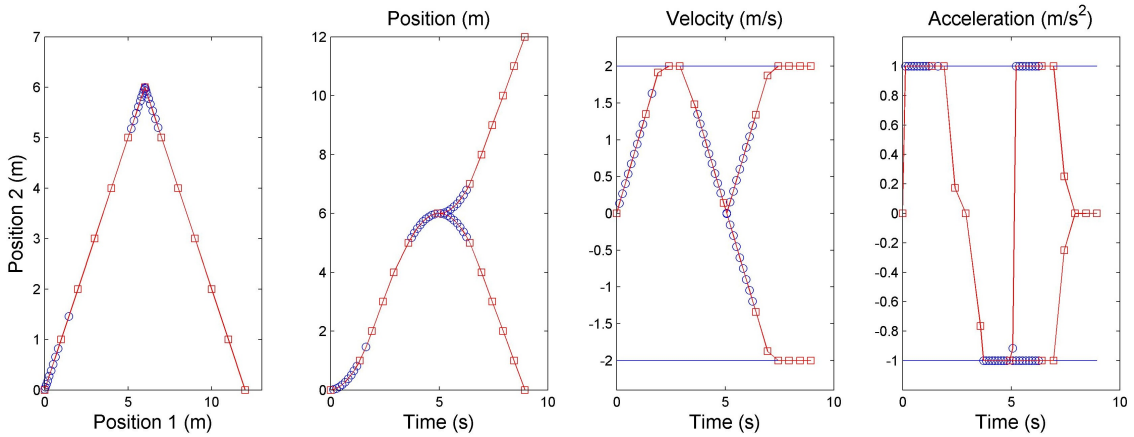


Figure 4.2.1: *In these four graphs, two joint values are plotted at the same time. The figure on the left shows the position of the two joints with respect to each other. One of the joints moves from 0 to 6 and then back to 0, while the other joint moves from 0 to 12 during the same time. The other three graphs show the position, velocity and acceleration of the two joints as functions of time. The squares mark the fixed points and the points marked by circles in the figure show the extra points that are added in order to more accurately model the dynamics of the problem. This can in the second figure be seen as a more smooth change of the position in time during the sharp turn.*

variables by one without changing the behaviour of the problem and at the same time it clears up the calculations. The equations of (4.1.1) are with these changes transformed into the kinetic equations that are used in the problem and that are stated in equations (3.3.1).

Using these new equations the acceleration goes down to zero when the maximum velocity is reached, as can be seen in figure 4.1.2. It is clear that the mechanisms that produced the oscillating behaviour now have been removed. A change in the velocity is now only dependent on a single acceleration value which means that the overcompensating that was seen before no longer can propagate through the solution.

4.2 Improved kinematics

In section 3.4 extra points were added to the problem between the fixed poses, in order to improve the kinematics. The effect of adding these extra points can be seen in figure 4.1.2 and perhaps even more clearly in figure 4.2.1. In both figures the additional points are marked with circles as opposed to the fixed points marked with squares.

Figure 4.2.1 shows two variables in the same plot. The example is set up to show what happens at a sharp turn. The path itself can be seen in the leftmost graph of figure 4.2.1 where the two variables are plotted against each other. The first variable is strictly increasing from 0 to 12 as was the case in the previous example. But the other variable changes direction mid-way, it increases from 0 to 6 and then decreases back down to 0 using all intermediate integer values as fixed positions. The extra points are added where they are most needed; at the beginning, where the object is accelerating up from the initial stand still, and close to the sharp turn, where the acceleration changes rapidly. The three remaining graphs show how the positions, velocities and accelerations of the two variables change in time. They show how the two variables follow exactly the same path leading up to the sharp turn and then break off into two symmetrical paths. The added extra points clearly show how the two variables slow down to a standstill right at the sharp corner and then accelerate in opposite directions.

4.3 The effect of adding energy

In section 2.6.4 an energy term was added to the objective function using the continuous forms of the variables, and in section 3.5 the discrete version of this was shown. The effect of adding a small energy term as a correction to the objective function can be seen when comparing the two top graphs with the two middle graphs in figure 4.3.1. In the top two graphs no energy term has been added but the middle two graphs show a case where the energy term is a slight correction to the total time in the objective function. The figure shows the position and acceleration of a simple two variable problem. One of the two identical variables has to move from zero to two with one as an intermediate fixed point. The other variable is set to move from zero to one and then back to zero. Both of the variables have to stand still at both the start and at the end of their movement. The fact that the second variable has to decelerate in order to make the turn back to zero, means that it is this variable that sets the constraints on how long time the movement takes. The first variable could complete its motion in less time, which means that there are multiple solutions to this problem when no energy term has been added.

Comparing the top two graphs with the middle two it is not immediately obvious what the difference is. The two graphs on the left showing the position look identical, but the two graphs on the right showing the acceleration differs slightly on one of the variables. The second variable is the one that switches back and forth between the maximum and minimum acceleration values. This is a unique solution and is not affected when a small energy term is added. However, the first variable is affected by the added energy term and among the set of solutions that go from 0 to 2 in the specified time the smoothest solution is selected. This can in the figure be seen as straightening out the small kinks in the original solution, most notably at the start and at the end where the original solution takes quite large jumps.

The energy term can also be allowed to play a more prominent role in the optimisation. The optimal solution to a problem that is only minimised with respect to time is always some kind of bang-off-bang solution. The acceleration will switch between the extreme values or be zero if the velocity constraints are limiting the movement. These instant changes of the acceleration might be unwanted when searching for a smooth and ergonomic solution. By adding an energy term to the objective function and using a larger constant c the bang-off-bang solution will be smoothed out. This can be seen in the bottom two graphs of figure 4.3.1 where the constant $c = 1$ is used, giving equal importance to the time and energy minimisations. The solution displays the same characteristics as before by switching between the extreme acceleration values. However, the transition is not instant; it smoothly changes from the maximum to the minimum value, spreading out the changes over a number of intermediate values. Looking at the graph showing the position - the bottom left graph of figure 4.3.1 - this is basically unchanged even if the motion takes a bit longer now that it is not time optimal.

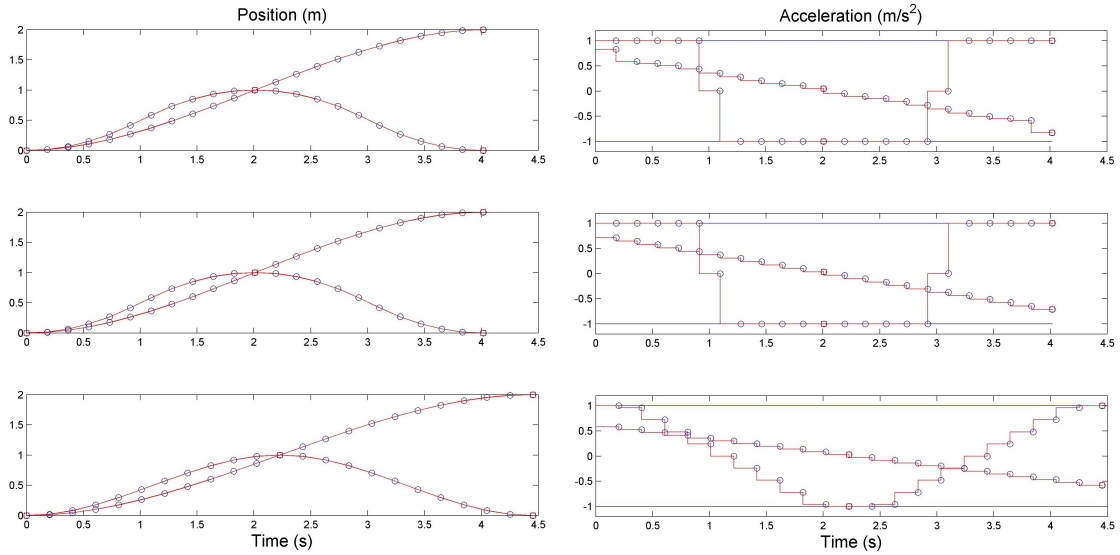


Figure 4.3.1: *The six graphs show how the relative importance of the energy minimisation affects the solution. For three different cases the position and acceleration of two joints are plotted as a function of time. The top two graphs show the solution when the energy term is not included, in the two middle graphs the energy term is only a small correction of the objective function and for the two bottom graphs the energy and time minimisation objectives are weighted equally.*

4.4 Comparing the approximation to the optimal solution

The fact that the acceleration in figure 4.3.1 is plotted as a discontinuous function of discrete steps is not only because it makes it possible to more clearly show the differences the energy term produces. It also shows the form of the selected mixed Galerkin solution space where the acceleration is a piecewise constant function. The figure can also be used to see how adding extra intermediate points gives a better approximation to the solution. Looking at the top case where no energy term has been added, the optimal solution for the second variable is to initially accelerate as much as possible and then at a specific point it should switch from maximum acceleration to minimum acceleration (or deceleration). It should also switch back to maximal acceleration at an exact point in time in order to come to a halt at the end of the motion.

The solution in the figure shows essentially this behaviour, but at each switch in the acceleration there is an intermediate value. This is because the optimal time for the switch does not match up to any of the added extra points. Instead the acceleration value for the interval containing the optimal switching point is selected so that the area under the graph (i.e. the velocity) is unaffected. Even if the extra points do not exactly match the optimal point, they narrow down the possible interval of the optimal time for the switch, thereby improving the approximation. Looking at figure 4.3.1 the first switch must be between the fifth and sixth points. Because of the engineered problem using simple values of the speed and acceleration limits, the first switching point can analytically be calculated to be at $t = 1$, right in between the fifth and sixth points.

4.5 Spatial constraints

In section 2.6.5 the theory of adding spatial constraints was presented. Instead of determining a vector of angles that the solution should contain, the spatial constraints specify a volume that a point of the manikin should visit. Figure 4.5.1 shows a two dimensional movement of a simple worm-like model with five identical joints. The path has been specified using four ball-constraints that the tip of the worm should visit in a specified order. It is possible to combine constraints so that the movement of both the joints and position is limited. However in this case the joint-constraints have all been removed so that it is only the four ball-constraints that specifies the movement.

The figure shows nine frames evenly spaced in time that starting from the top left shows the motion from left to right. The solution is time optimal and the velocity and acceleration limits are identical for all the joints. This can be seen in the figures as a desire to use as many joints as possible at the same time. It can for example be seen in the first four or five frames that describe the motion between the first and second ball. Here the three outermost joints perform almost exactly the same movement, adding as much speed as possible to the tip of the worm.

The smooth curve that just touches each of the target balls shows the path that the tip of the worm follows through the motion. The other curve that in arcs goes through the centre of all balls shows how the tip would move according to the initial linearly interpolated path. The turns for this path are so sharp that the worm would have to come to a complete halt to be able to follow the path. It is clear that the new path is faster since smoothing out the corners makes it possible to keep some speed throughout the movement.

The fact that it is only a single point - the tip of the worm - that has to fulfil the constraints gives the model a lot of freedom. In the case of the manikin this freedom can cause problems. The final solution should make use of the initially calculated poses that assures that the motion is collision free. If only single points are used to fix the model the final solution can deviate very much from the initial pose, remember the discussion about multiple poses in section 1.2 and illustrated in figure 1.2.2. However, it is good to give the model some freedom when optimising the movement in order to find a fast and smooth movement. By adding the same type of ball-constraints for all the rigid bodies that make up the model the final solution can not deviate very much from the initial poses, but the model is given some additional freedom since small deviations around the initial pose are allowed. Setting up the constraints in this way makes it easy to use the clearance of the different joints. Points that are allowed to move around quite freely can be used to optimise the solution while other joints might be close to fixed objects forcing the final solution to closely follow the initial path.

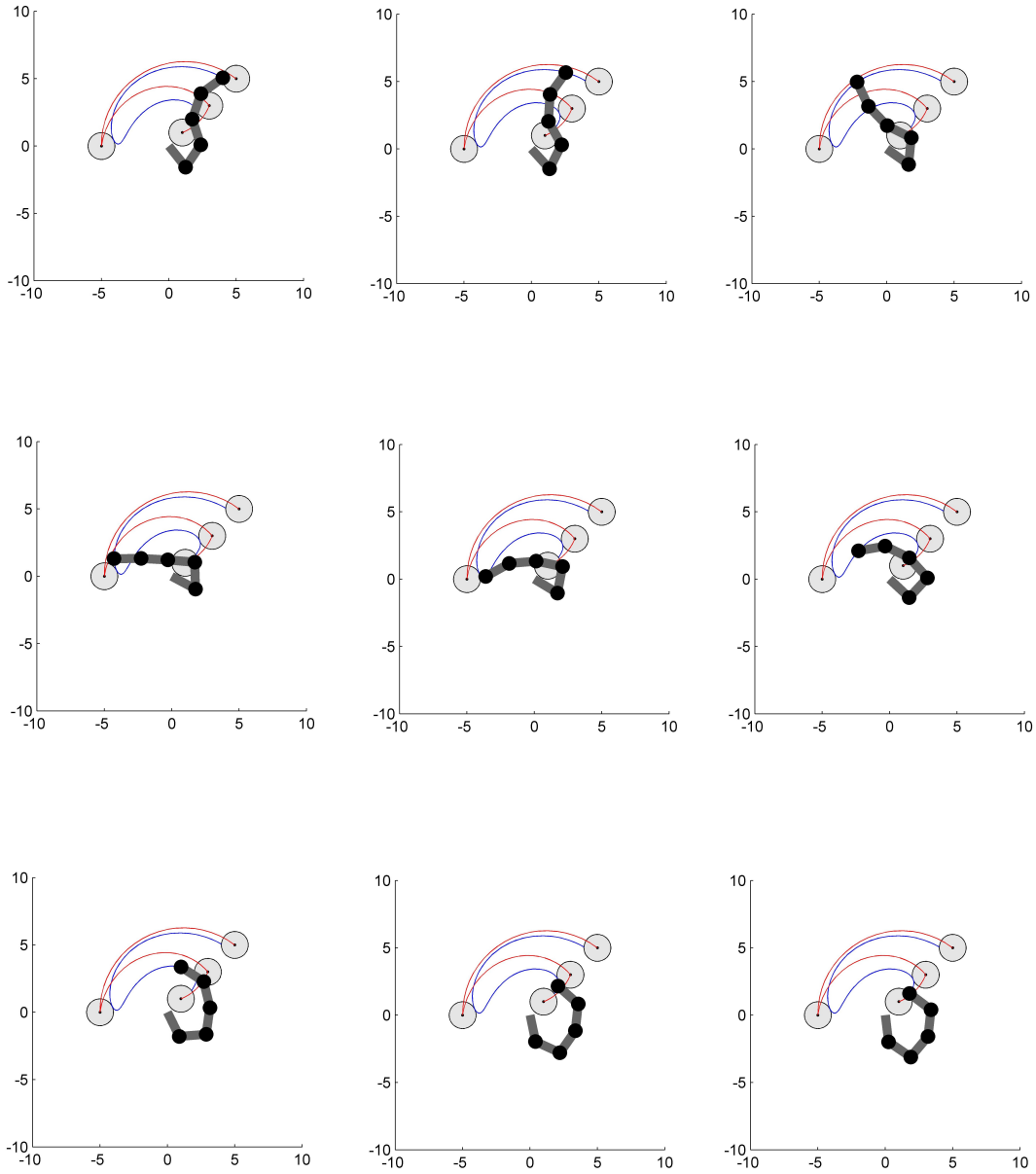


Figure 4.5.1: *These nine figures show the movement of a simplified, two dimensional model. In this worm-like structure the model consists of five rotational joints with the base joint fixed to the origin. From left to right, starting at the top left, the figures show snapshots of the movement that are evenly sampled in time. The circles represent the target areas that the tip of the model (the TCP) should visit in a specified order. The movement of the TCP is also shown as a plotted curve. The other curve that consists of arches through the centre of all circles shows the TCP movement of the corresponding linearly interpolated curve.*

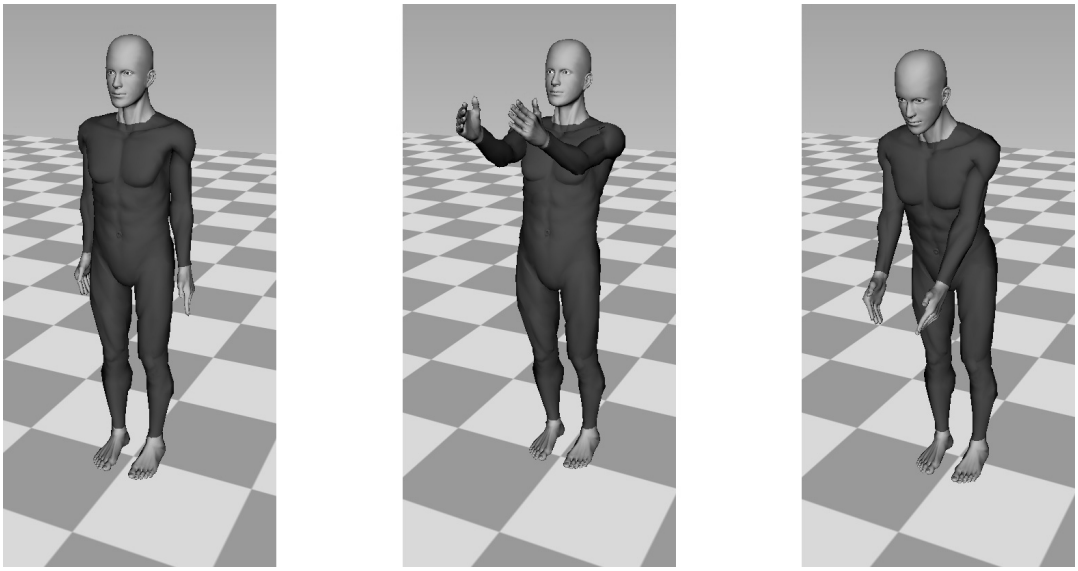


Figure 4.6.1: *The figures show three poses from left to right that make up a path for the manikin to follow.*

4.6 Manikin results

The cases seen in sections 4.1 and 4.2, where the spatial constraints have not yet been added, find an optimal solution only using the angles as variables. When transferring these simple problems to the case of the manikin it is only a matter of scaling up the problem. All joint values are set up identically in the optimisation problem and reading joint values directly from the manikin, for all poses making up the path, gives the input for the optimisation problem. Figure 4.6.1 shows a short path made up of three fixed poses that the manikin should follow, but it is hard to show the solution to a problem like this only using figures. The objective is to find an optimal motion which means that the motion will most often not deviate too much from the linear solution; the main difference that can be seen is how the manikin accelerates and decelerates as opposed to the constant speed solution for the linear case.

Even after adding spatial constraints the found solution generally follows the initial path closely. It is necessary to keep the deviations from the initial path small to ensure that the conditions of ergonomics and balance are still fulfilled. This is done by having a short radius on all the added ball-constraints. If the radius is set to a bigger value it is easy to see how the optimisation algorithm exploits this freedom. An example of this can be seen when comparing the initial path in figure 4.6.1 with the solution in figure 4.6.2. The constraints are set so that the centre of mass of each rigid body in the manikin is allowed to deviate at most 5 cm from the initial path. In order to show the behaviour clearly, no constraints have been placed on the joint values. The series of frames showing the solution in figure 4.6.2 shows how the manikin twists its upper body, using the many joints in its spine to as quickly as possible reach the final pose. It is clear that even if the final pose of the solution only deviates 5 cm from the final pose in the initial solution, the ergonomics are not even close to being fulfilled. Setting more strict spatial constraints and also adding constraints on the joint values guides the manikin into a more ergonomic final path.

The optimisation problem becomes very much larger when the ball-constraints are added which means that the running time is greatly affected. The interior point algorithm is more efficient when it is given an interior point as initial value. An interior point means that it is a feasible point that does not break any of the constraints. The good thing is that the final solution calculated without ball-constraints will always be a feasible point. The points will always lie in the centre of the ball which means that the solution is a valid solution even when the ball-constraints are added. This also means that the found time values are also valid and they effectively set an upper limit on how long time the final movement can take. The fact that the simple problem always gives a valid solution makes it worthwhile to always calculate this solution and give it as a starting point for the

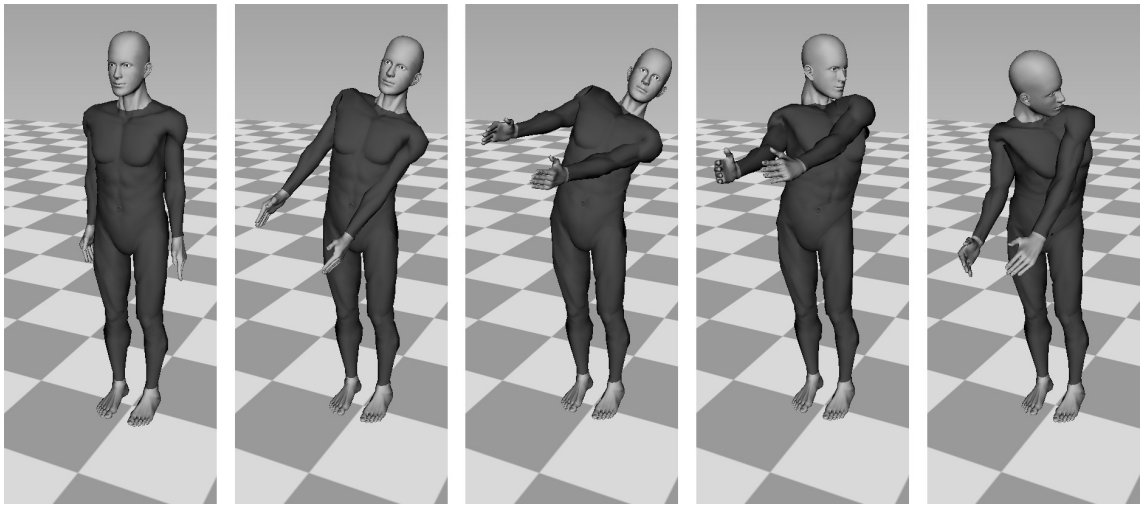


Figure 4.6.2: *Giving the manikin too much freedom can produce strange results. The five figures shows a manikin motion where all joints are allowed to move a maximum of 5 cm from their initial position. The initial path is defined by the three poses of figure 4.6.1.*

optimisation using ball-constraints.

The ball-constraints gives the manikin extra freedom which means that the solution will always be faster than the original problem. This can be seen in table 4.6.1 where the final time of the solution is shown for the same problem, using a number of different sizes of the ball-constraints. The effect on the final time is quite large, but the found solution might not be a desirable one. For example the case with $r = 5$ is the same solution as the one shown in figure 4.6.2.

Table 4.6.1: The table shows how the final time of the solution decreases as the manikin is given more freedom in the movement. The radii r determines how much each point is allowed to deviate from the original path.

Radii	$r = 0$ cm	$r = 1$ cm	$r = 5$ cm	$r = 10$ cm
Final time of the solution	2.5858 s	1.0083 s	0.6305 s	0.4871 s
Fraction of the original solution time	100 %	39 %	24 %	19 %

Even without ball-constraints the amount of memory that is allocated during the optimisation can become a problem. Table 4.6.2 contains data from an optimisation of a rather large problem with 255 fixed poses. As can be seen in table 4.6.2 there are also 1270 additional points; 5 between every fixed pose. The table also shows the memory usage, it is especially the number of nonzeros in the equality constraint jacobian that can cause problems as the number of variables and constraints grow. The size of the optimisation problem also causes the running time to go up, from the table we can see that it is the time spent in the IPOPT optimisation code that has the largest effect on the running time.

Table 4.6.2: The table shows data from a large optimisation using 255 fixed poses and 5 additional points between each fixed pose.

Problem setup	
Number of fixed poses	255
Total number of extra points	1 270
Total number of variables	699 608
Number of equality constraints	493 776
Number of nonzeros in equality constraint Jacobian	2 139 048
Number of nonzeros in Lagrangian Hessian	781 488
Problem evaluation	
Number of iterations	52
Number of objective function evaluations	73
Number of objective gradient evaluations	53
Number of equality constraint evaluations	73
Number of equality constraint Jacobian evaluations	53
Number of Lagrangian Hessian evaluations	52
Total CPU secs in IPOPT (without function evalutions)	2787.008
Total CPU secs in NLP function evaluations	16.579

5 Conclusions and Future Work

The aim of this thesis, which was to improve the motion generation of the manikin, has been reached. The manikin can move along a pre-defined path without breaking velocity or acceleration constraints. It also does this in a smooth and efficient manner - and the energy term in the objective function can be used to weigh the two objectives smooth/efficient against each other. Additional functionality such as spatial constraints have also been added making it possible to include collision avoidance in the optimisation.

It is also important to see this project as a basic framework where optimisation methods, objective functions and variable constraints are easy to replace or develop further. An example of this is the Galerkin method which is used to obtain numerical approximations to the sought functions. This could be replaced with a pseudo spectral method that does not discretise evenly in time but adds points where they are most needed in order to get accurate approximations of the functions.

The fact that the motion generation is a post processing stage that is used after the calculation of the actual path, does cause some problems and some of them are not yet fully resolved. We have seen that adding the spatial ball-constraints to the manikin and optimising for time without including ergonomics can give very strange solutions. It is then a trade-off between giving the model as much freedom as possible in order to find a smooth and efficient path, while still constraining the variables so much that the final solution still fulfils the balance and ergonomic conditions.

One way to get around this could be to include some kind of measure of the ergonomics in the optimisation. Another could be to closely connect the ergonomic conditions to the constraints in the optimisation without strictly including them.

It might seem a bit counterproductive to include more and more of the preprocessing step into the optimisation. And it could well be that the easiest way to overcome the problems that arise by dividing the problem into a preprocessing and a postprocessing stage is simply to combine the two steps. The downside to this is of course that the running time would be affected. Perhaps it is not always necessary to calculate the movement in time, it can also be useful to calculate a series of valid poses that can then be used in different ways - not only to produce a movement but the ergonomics of the poses themselves could to some extent be examined without having the full motion.

One improvement that has been looked into is the matter of including full dynamics. In that case the rigid bodies are required to be in dynamic equilibrium, measuring forces and moments instead of accelerations. Full dynamics could give more realistic movements for the manikin, getting away from the static equilibriums that are required at present for every pose.

There are other improvements that could be made, improving the project whether it was merged with the preprocessing stage or not. One of these improvements is to obtain actual velocity and acceleration limits from testing. This might be done by studying recordings of human motion or using approximations of muscle strength in order to calculate acceleration limits. Since it might be easier to measure muscle forces this could be another argument for using the moments and forces as control signals.

Yet another possibility for generating realistic motions is to try to mimic human behaviour using some kind of machine learning. This is not something that has been considered for this thesis and is a solution that lacks the generality of the presented solutions and makes it hard to use the generated motion to examine for example ergonomics. But it could perhaps be a good method if the goal simply is to generate feasible human motions.

There are of course a lot of areas that require further work but this thesis has improved the visualisation of manikin motions and has at the same time provided a good general optimisation framework. The path of the manikin can be optimised with respect to both energy and time, and the movement can at the same time be limited using a number of different constraints. These constraints are implemented in a modular way, making it easy to include or replace constraints along with their respective derivatives. The constraints that are implemented are used to construct smooth and efficient paths which improve the visualisation of the movement through the found valid poses. This is an important part of the manikin project since the credibility of a calculated movement is often dependant on how well it can be visualised.

References

[1] IPS. *Industrial Path Solutions*. 2012. URL: http://www.industrialpathsolutions.com/.

[2] Peter Blanchonette. "Jack Human Modelling Tool: A Review". In: (2010). Available at Austrailian Government, Department of Defence, Defence Science and Technology Organisation (DSTO) Scientific Publications Online. URL: http://hdl.handle.net/1947/10032.

[3] Human Solutions. *Human Solutions: RAMSIS*. 2012. URL: http://www.human-solutions.com/automotive/index_en.php.

[4] Dassault Systèmes. *Dassault Systèmes: Human Modeling*. 2012. URL: http://www.3ds.com/products/delmia/solutions/human-modeling/overview/.

[5] Niclas Delfs. "Manikin in Time; Development of a Virtual Manikin with Inverse Kinematics and Comfort". MA thesis. University of Gothenburg, 2011.

[6] Niclas Delfs. "Manikin in Time; Development of the Virtual Manikin with External Root and Improved Inverse Kinematics". MA thesis. Chalmers University of Technology and University of Gothenburg, 2012.

[7] Kenneth Eriksson, Don Estep, Peter Hansbo, et al. *Computational Differential Equations*. Studentlitteratur, Lund, 1996.

[8] Krister Svanberg. *Linjär Optimering*. Optimeringslära och Systemteori, Institutionen för Matematik, Kungliga Tekniska Högskolan.

[9] Andreas Wächter and Lorenz T. Biegler. "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming". In: *Mathematical Programming* 106.1 (2006), pp. 25–57. URL: http://www.springerlink.com/content/r31116mp70171220/.

[10] Yoshiaki Kawajir, Carl Laird, and Andreas Wächter. *Introduction to Ipopt: A tutorial for downloading, installing, and using Ipopt*. 2010. URL: http://www.coin-or.org/Ipopt/documentation/.

[11] Timur Ustyan and Viktor Jönsson. "Implementation of a generic Virtual Robot Controller". MA thesis. Chalmers University of Technology, 2011.