ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bio-Inspired Robotics Lab
Prof. Dr. Fumiya Iida

Semester Project

Optimizing dynamic motions of two-link pendulum

Spring Term 2011

Supervised by:

Fumiya Iida

Author:

Christos Lataniotis

Contents

Abstract

The problem of optimizing dynamic motions of a double pendulum is considered. The goal is to find an optimal control strategy that will drive the system from given initial to given final conditions in an optimal way. Here the optimality was expressed in terms of the effort of the joint motors, although different optimality criterions can be easily used instead. Since no prior knowledge exists about the time required for this transition and the path that has to be followed the control inputs have to be optimized together with the trajectory that has to be followed. By parametrizing the joint trajectories using cubic B-splines and applying some fixed control strategy (feedback linearization and LQR control) this optimal control problem is reduced to a relatively low dimensional parameter optimization problem. The algorithm is tested on two tasks, one resembling a 'shoulder' movement from a horizontal to a vertical posture and one resembling a throwing-like motion. It is shown that by using this approach, swinging motions can be obtained that can significantly reduce the effort of the joint motors in order to achieve these tasks. Additionally, the effectiveness of the approach is shown in case of imposing angular velocity and/or torque constraints.

Chapter 1

Preliminarities

1.1 Problem Definition

On this project the case of a two-link pendulum (or double pendulum) is considered. This is a thoroughly investigated non-linear dynamic system. As shown, in Figure

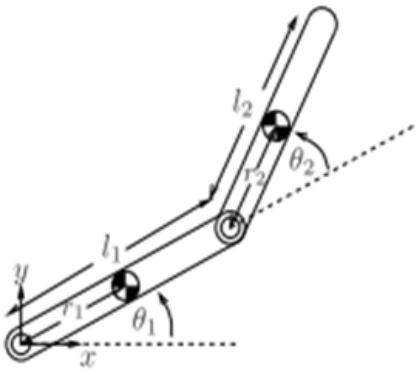

Figure 1.1: The double pendulum.

1.1 it is composed of two rigid bodies connected with rotational joints. In our case each rotational joint is actuated with a motor. In Appendix A, the equations of motion for this system are derived in detail. On this point the general form of the equations is considered([2]):

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_v\dot{q} + F_d(\dot{q}) + G(q) + \tau_D = \tau \tag{1.1}$$

where $q \in \mathbb{R}^n$ is the joint variable vector, $M(q)$ the mass matrix, $C(q,\dot{q})$ the Coriolis/centripetal vector, $F_v\dot{q}$ the viscous friction, $F_d(\dot{q})$ the dynamic friction, $G(q)$ the gravity, τ_D the disturbances and/or other unmodelled dynamics and τ the externally applied generalized forces (force for prismatic joint and torque for rotational joint). It should be noted that is the general (closed-)form of the equations of motion of any robotic arm with rotational/prismatic joints. For the following analysis friction terms and disturbances are going to be neglected, thus equation 1.1 becomes

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau \tag{1.2}$$

Now, for a double pendulum with dynamic behaviour as described in 1.2, we want to achieve a transition from an initial state $q_0 = \begin{bmatrix} \theta_{0,1} & \theta_{0,2} & \dot{\theta}_{0,1} & \dot{\theta}_{0,2} \end{bmatrix}^T$ to a

final state $q_f = \begin{bmatrix} \theta_{f,1} & \theta_{f,2} & \dot{\theta}_{f,1} & \dot{\theta}_{f,2} \end{bmatrix}^T$ by controlling the torques applied on each joint. Most importantly, the goal is to try to find such torques that not only achieve this transition but the *effort*,i.e. the energy consumed by the motors is as low as possible. Therefore, we are trying to solve an optimization problem roughly described as follows

$$\tau^* = argminJ$$

Subject to:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau$$

$$q(0) = q_0, q(t_f) = q_f$$

(additional physical constraints)

We are looking for the torque sequence $\tau^* = \begin{bmatrix} \tau_1^* & \tau_2^* \end{bmatrix}^T$ that minimizes some performance criterion (objective function) J, subject to the constraints of the system dynamics, initial and final desired conditions and maybe additional constraints that are going to be considered later, namely angular velocity bounds for each joint and motor torque bounds. We can see that this optimization problem is not trivial and still the problem is not clearly defined. The analysis that follows on this chapter is summarized below

- Define a control strategy to be followed.
- Describe the manipulator trajectory from initial to final state.

Then in Chapter 2 we are going to put our findings together and rewritte the optimization problem.

1.2 Controlling the pendulum

1.2.1 Introduction

The general (closed) form of a robot arm equation (equations 1.1,1.2) was given previously. The purpose of this section is to figure out a strategy of calculating the torque that each motor should produce ($\tau_i(t)$) in order to follow a desired trajectory, $q_d(t)$. First tbe concept of feedback linearization is going to be discussed, since this is a standard technique in controlling non-linear systems. Then since, some sort of optimal choice - and even better with low computational effort- of torques would be nice, the concept of Linear Quadratic Regulator is going to be discussed and finally applied on our problem.

1.2.2 Computed Torque Control - Feedback Linearization

On this point it is supposed that a desired trajectory $q_d(t)$ has been selected for the arm motion. In order to ensure trajectory tracking by the joint variables, we define the *tracking error* as follows:

$$e(t) = q_d(t) - q(t) \tag{1.3}$$

It is straightforward to show that

$$\dot{e}(t) = \dot{q}_d(t) - \dot{q}(t), \ddot{e}(t) = \ddot{q}_d(t) - \ddot{q}(t)$$

If we solve for $\ddot{q}$ the equation of motion 1.2 yields

$$\ddot{q} = M(q)^{-1}(-C(q,\dot{q})\dot{q} - G(q) + \tau)$$

and by using the definition of tracking error (1.3) follows that

$$\ddot{e} = \ddot{q_d} + M(q)^{-1}(C(q,\dot{q})\dot{q} + G(q) + \tau_d - \tau) = \ddot{q_d} + M(q)^{-1}(N(q,\dot{q}) + \tau_d - \tau) \tag{1.4}$$

Now, two functions are defined. The control input function

$$u = \ddot{q_d} + M(q)^{-1}(N(q,\dot{q}) - \tau) \tag{1.5}$$

and the disturbance funtion

$$w = M(q)^{-1}\tau_d \tag{1.6}$$

Now, by using the tracking error dynamics 1.4 and plugging in the definitions 1.5 and 1.6 we get the following tracking error dynamics equation

$$\frac{d}{dt}\begin{bmatrix} e \\ \dot{e} \end{bmatrix} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}\begin{bmatrix} e \\ \dot{e} \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} u + \begin{bmatrix} 0 \\ I \end{bmatrix} w \tag{1.7}$$

Equation 1.7 describes the dynamics of the tracking error (i.e. how the tracking error evolves over time) with respect to the control input u and the disturbance function w. We can see that this equation describes a linear, time-invariant system. This means that choosing a stabilizing u for 1.7 is much easier than choosing a stabilizing τ for 1.4. It should be noted that, given some stabilizing u for 1.7 the corresponding torque can be calculated by

$$\tau = M(q)(\ddot{q_d} - u) + N(q,\dot{q}) \tag{1.8}$$

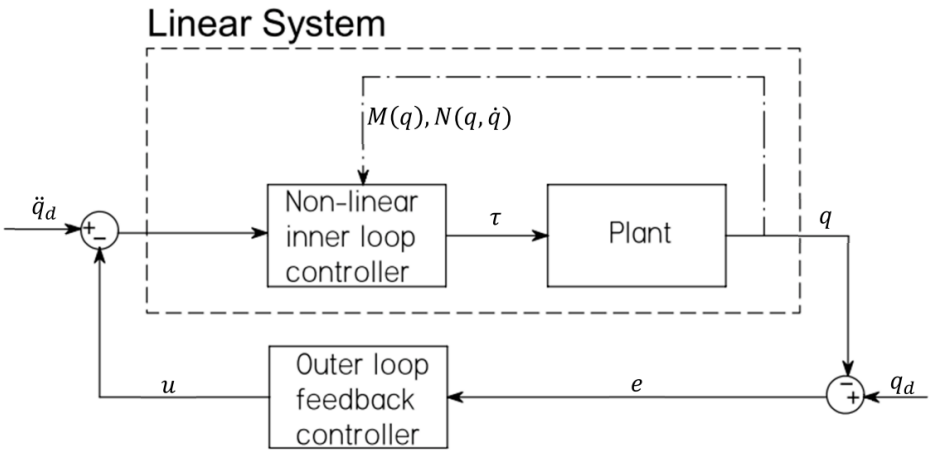

Figure 1.2: The computed-torque control scheme.

This is known as the *computed-torque control law*. The importance of this approach is therefore the fact that a complicated non-linear control problem is transformed to a simple design-problem of a linear system. The resulting control scheme is shown in Figure 1.2.
We can see that the control scheme consists of an inner non-linear loop (that produces τ, using 1.8)plus an outer control loop (that produces u for stabilizing 1.7). It is important to note that the major weakness of this approach is its dependance on our knowledge of the model (M, N). Especially matrix N involves difficult to exactly calculate terms, like friction. Therefore a good identification of the system is required in order to apply this approach (this is outside the scope of the current project and for the following analysis it is assumed that the system model is sufficiently well known).
The next step would be to define a control strategy for calculating u of the outer control loop. There are many different approaches, here the LQR control scheme was selected, which is going to be presented next.

1.2.3 LQR Control

Recall that the outer control loop system (1.7) is a Linear Time-Invariant system which is generally written in the form:

$$\dot{x} = Ax + Bu \tag{1.9}$$

with $x(t) \in \Re^n, u(t) \in \Re^m$ (in the fully actuated double pendulum case $n = 4, m = 2$).
We want to compute the state feedback gain, K, in

$$u = -Kx \tag{1.10}$$

so that the closed-loop system (which is derived by using 1.10 in 1.9):

$$\dot{x} = (A - BK)u \tag{1.11}$$

is asymptotically stable. A system is said to be (globally) asymptotically stable if and only if for any initial condition the states are 'finally' going to zero.

Apart from achieving the goal of stabilizing the system it is desirable to not use much control energy for this transition (which practically makes sense). These requirements for u can be formulatted using the following *quadratic performance index*:

$$J = \frac{1}{2}\int_0^\infty \left(x^T Q x + u^T R u\right) dt \tag{1.12}$$

where $Q \in \Re^{n\times n}$ is a symmetric positive semidefinite matrix (denoted $Q \succeq 0$) and $R \in \Re^{m\times m}$ is a symmetric positive definite matrix ($R \succ 0$). The objective is then to find the gain matrix, K, such that when the control law 1.10 is applied to the system 1.10, the corresponding value of the quadratic performance index J, is minimum. [1] Once an optimal gain has been determined it is *guaranteed* that all the states and control inputs of the closed-loop system go to zero in finite time. The existence of an optimal K requires a certain *finite* value of the performance index J. This means that the infinite integral in 1.12 is finite, therefore the function $x(t)^T Q x(t) + u(t)^T R u(t)$ goes to zero as time t increases. By using the definition of 2-norm we can see that:

$$x(t)^T Q x(t) = \left\| \sqrt{Q} x(t) \right\|_2^2$$
$$u(t)^T R u(t) = \left\| \sqrt{R} u(t) \right\|_2^2$$

with the square root of a matrix defined as $A = \sqrt{A^T}\sqrt{A}$. Since these norms go to zero over time and $R \succ 0$ (i.e. R cannot be a zero matrix) both functions $\sqrt{Q}x(t)$ and $u(t)$ go to zero. Under the assumption that $(A, \sqrt{Q})$ is observable then x(t) also goes to zero. [2]
The solution of this problem is a standard result in control theory and drops down to

$$K = R^{-1} B^T P \tag{1.13}$$

where $P \in \Re^{n\times n}$ is a symmetric matrix that is the solution of the so-called *Riccati equation*:

$$A^T P + PA + Q - PBR^{-1}B^T P = 0 \tag{1.14}$$

It is worth to notice that the optimal LQR feedback gain is a function of the system characteristics (matrices A,B) and the design variables Q,R. Roughly speaking higher values of R compared to Q results to more energy efficient solution with

[1] The definition of a positive semidefinite matrix Q($\in \Re^{n\times n}$), is

$$Q \succeq 0 \Leftrightarrow x^T Q x \geq 0\ ,\ \forall\, x \in \Re^n$$

and similarly $x^T Q x > 0\ ,\ \forall\, x \in \Re^n\ ,\ x \neq 0_n$ for Q positive definite. Moreover, if $\lambda \in \mathbb{C}$ is an eigenvalue of Q, and $y \in \Re^n$ the corresponding eigenvector, then by the definition of the eigenvalue of a matrix follows :

$$\lambda y = Qy$$
$$\lambda y^T y = y^T Q y, \lambda \left\|y\right\|_2^2 \qquad\qquad = y^T Q y$$

Where $\|\cdot\|_2 \geq 0$ is the 2-norm. We can clearly see that for positive semidefinite Q the real part of each eigenvalue has to be greater or equal to zero and for positive definite Q greater than zero.

[2] For proof of this proposition the interested reader can refer to classical control textbooks, like T. Kailath, *Linear Systems*, Prentice-Hall, 1980.

higher state response time and vice versa. The solution of the Riccati equation and of the LQR problem in total can be easily solved nowadays using standard routines in Matlab and other software packages.
As it was shown earlier (Figure 1.2) the control input u, is not the actual input to the robotic arm. The actual input of the arm is the torque τ which is related to u by the (non-linear) relation 1.8. Therefore it has to be checked whether there still exists a link between the energy minimization of u (using LQR) and the energy minimization of the actual input to the arm, τ. We take the norm (does not matter which) of each side in 1.8 and use norm inequalities:

$$\begin{aligned}\|\tau\| &= \|M(q)(\ddot{q_d} - u) + N(q, \dot{q})\| \\ &\leq \|M(q)\|\|(\ddot{q_d} - u)\| + \|N(q, \dot{q})\| \\ &\leq \|M(q)\|\|\ddot{q_d}\| + \|M(q)\|\|u\| + \|N(q, \dot{q})\|\end{aligned}$$

Thus, by keeping $\|u\|$ small it would be expected to make $\|\tau\|$ smaller but a formal statement cannot be made without knowing $\|M(q)\|$ and $\|N(q, \dot{q})\|$. Since the energy is not formally minimized using this approach the resulting energy consumption should be considered as *suboptimal*. [3]

[3] An interesting extension of the current work would be to test an optimal control approach weighting directly $e(t)$ and $\tau(t)$ in the performance index *[9]*.

1.3 Describing a manipulator trajectory

1.3.1 Introduction

For the execution of a specific robot task, some sort of motion planning has to take place. The goal of this motion planning or trajectory planning is to generate the reference inputs to the motion control system which ensures that the manipulator executes the planned trajectories. As noted in [1], the difference between *path* and *trajectory* should be explained first, in order to avoid confusion. A path denotes the locus (or via) points in the joint space (or operational - Cartesian space) [4]. On the other hand, a trajectory is a path on which a timing law is specified, i.e. not only we define the points that the manipulator has to pass through but also when it should pass from each of them. This also implies that velocities and accelerations are also implicitly specified. Here we focus on the later case, of defining a trajectory, for reasons that will be discussed in the next Chapter. In the following analysis the trajectory is described in terms of the joint space, because mapping the Cartesian space of the end effector to the joint space can complicate the problem and moreover in our case we do not have some sort of motivation for working on the operational space (end effector constraints, specific via points, etc).
The following assumptions are made:

- No spatial constraints exist, i.e. the pendulum can move freely over its entire workspace without having to 'worry' about any obstacle.
- We do not require any specific via point to belong on the desired trajectory. We only care about the initial and final conditions.
- The transition from the initial to the final state begins from time $t_0 = 0$ and ends at time t_f.The final time t_f is a parameter to be found and no specific value is known a-priori.

There also some general characteristics that a joint space trajectory planning algorithm is expected to have([1]):

- The generated trajectories should not be very demanding from a computational viewpoint.
- The joint positions and velocities should be continuous functions of time (also continuity of accelerations is also important since it implies more smooth changes to the output torques of the motor)
- Smoothness of the trajectory (continuous derivative) should hold as much as possible.

The desired trajectory is defined as

$$q_d(t) = \begin{bmatrix} \theta_{d,1}(t) & \theta_{d,2}(t) & \dot{\theta}_{d,1}(t) & \dot{\theta}_{d,2}(t) \end{bmatrix}^T \tag{1.15}$$

with $\theta_{d,i}(t)$ the desired angle trajectory of joint i, and $\dot{\theta}_{d,i}(t)$ the desired anglular velocity trajectory of joint i.
This function is usually determined by some given points in the joint space, like initial / final conditions or via points that have to be followed. The simplest way

[4] Joint space is a coordinate system used to describe the state of the robot in terms of it's joint states. Operational space refers to a Cartesian coordinate system attached to the robot's end effector. One can switch from one representation to the other by using the *kinematics* to get from joint space to operational space and the *inverse kinematics* for the opposite. A caveat is that in the case of inverse kinematics uniqueness of solution is not guaranteed in general.

to interpolate these points is by using (high order) polynomials. Such approaches used to be popular in earlier industrial robotics applications due to the simplicity of the solution. Even though such methods can provide an accurate fit of position, velocity and acceleration profile at the given points, the need to constrain derivatives demands a higher degree of the approximating polynomial. Also, as the degree increases, so does the tendency for the polynomial to oscillate wildly between the interpolated(given) points. Another disadvantage of using polynomials to approximate a trajectory is that a local modification to the trajectory requires a complete re computation of all the polynomial coefficients; this makes trajectory optimization harder, especially in case it is taking place on-line.
The problems resulting from the use of high-order polynomials can be avoided by fitting successive low-order polynomials (such as cubics) to successive groups of data. The resulting piecewise polynomial function is continuous, but has discontinuous derivatives at curve segment intersections which gives significant acceleration jumps at those points. Therefore, it is desirable, to make the desired trajectory $q_d(t)$ continuous up to the 2^{nd} time derivative. This can be achieved by using cubic splines. Higher order splines would also make sense to use. However, one common limitation of these splines techniques is that a local modification to any region along the trajectory requires the re computation of the entire trajectory. In order to overcome this limitation B-Splines are used which are going to be discussed next.

1.3.2 B-Splines

B-Splines of order k, are $k-1^{th}$ order piecewise C^{k-2} continuous polynomials. For example, for cubic splines (which are selected in our case), $k=4$, the piecewise polynomials are of order 3 and C^2 continuity is guaranteed, i.e. continuity up to the 2^{nd} derivative. Assume that the trajectory of one joint of the double pendulum(q(t)) is going to be described using B-Splines. Then

$$q(t,P)=\sum_{j=0}^{m} p_j B_{j,k}(t) \tag{1.16}$$

where $P=[p_0 \dots p_m]^T$ is the control points vector and $B_{j,k}$ is the B-Spline basis function (with C^{k-2} continuity). For simplicity reasons here we silently assumed that $q(t,P)\in\Re$ but similar analysis can be done for the more general case $q(t,P)\in\Re^n$. In this case, elements p_j are actually the y-coordinates of the control points, therefore in order to fully define them in the trajectory space we need the corresponding x axis coordinates. The knot vector $T=[t_0 \dots t_{m+k}]^T$ corresponds to these x coordinates, but using k additional elements compared to vector P (this will be explained later). The elements of t should be in ascending order, i.e. $t_0 \le t_1 \le \dots \le t_{m+k}$ and in case of *uniform* B-Splines (which are going to be used here) the elements are equally spaced, i.e. $t_1-t_0=\dots=t_{m+k}-t_{m+k-1}$.
The Basis function, $B_{j,k}$ can be computed using the Cox-de Boor recursion formula([5]):

$$B_{j,k}(t)=\frac{t-t_j}{t_{j+k-1}-t_j}B_{j,k-1}(t)+\frac{t_{j+k}-t}{t_{j+k}-t_{j+1}}B_{j+1,k-1}(t) \tag{1.17}$$

starting with

$$B_{j,1}(t)=\begin{cases}1 & if\ t_j \le t < t_{j+1}\\ 0 & otherwise\end{cases} \tag{1.18}$$

In Figure 1.3 the basis functions are shown from order 1 up to order 3, by applying the recursive formula. Higher order basis functions maintain the bell-shape that is observed in the quadratic case.

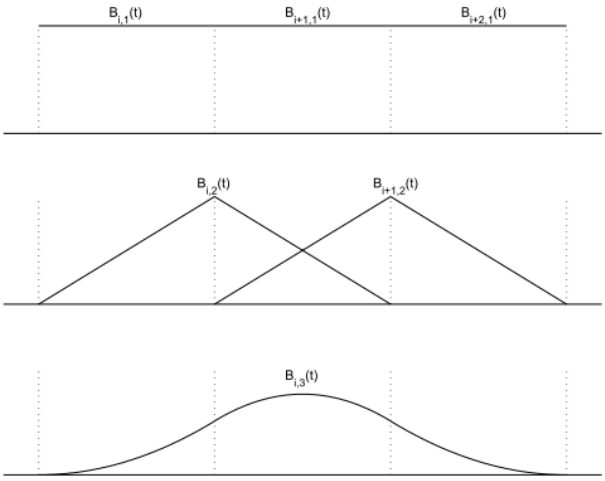

Figure 1.3: A quadratic B-Spline produced by recursion.

From 1.16 we can see that the final B-Spline curve is produced by a weighted sum of the Basis functions;control points act as the weights. An example is shown in Figure 1.4.

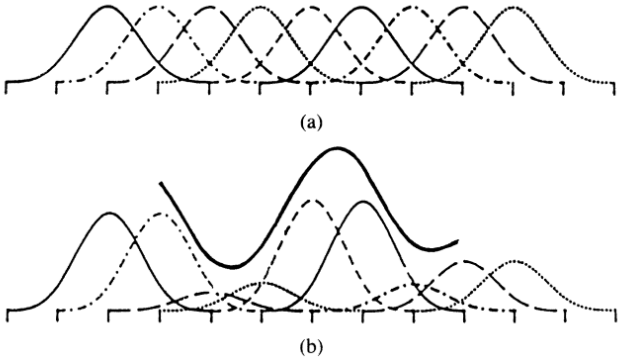

Figure 1.4: (a) The Basis functions centered on every knot point. (b) The final form of each basis function after being weighted by the corresponding control point and the resulting B-Spline curve.

1.3.3 Cubic B-Splines in parametric form

Now we are focusing on the B-Splines of order $k = 4$ (cubic). It is convenient to introduce an alternative way of describing the B-Spline, known as *parametric form*. First the variable u(t) is defined as follows:

$$u(t) = \frac{t - t_i}{t_{i+1} - t_i} \ , \ u \in [0, 1] \tag{1.19}$$

After some calculations equation 1.16 becomes:

$$q(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \end{bmatrix} \tag{1.20}$$

The vector $\begin{bmatrix} p_{i-2} & p_{i-1} & p_i & p_{i+1} \end{bmatrix}^T$ in 1.20 refers to four consecutive control points. The indices can be chosen differently (for example start from i-1) as long as the four control points remain consecutive.

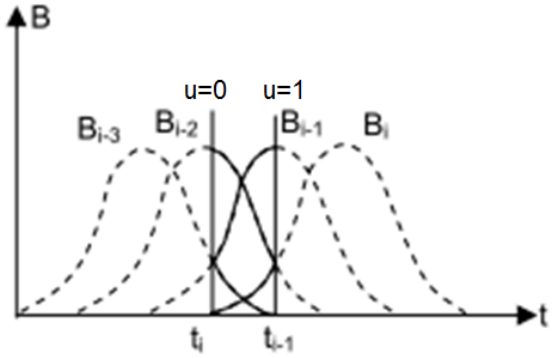

Figure 1.5: Basis functions that affect $q(u)$ for $u \in [0, 1]$

For every time, t, the corresponding value of u is calculated using 1.19. Then q(u) is calculated from 1.20. As shown in Figure 1.5 only four neighbouring control points will affect the final B-spline curve for every value of t. We can also see that in order to fully define the B-Spline over a region we need 4 Basis functions 'acting' in the whole region of interest, therefore, depending on the numbering in the vector $\begin{bmatrix} p_{i-2} & p_{i-1} & p_i & p_{i+1} \end{bmatrix}^T$ in 1.20. One example is shown:

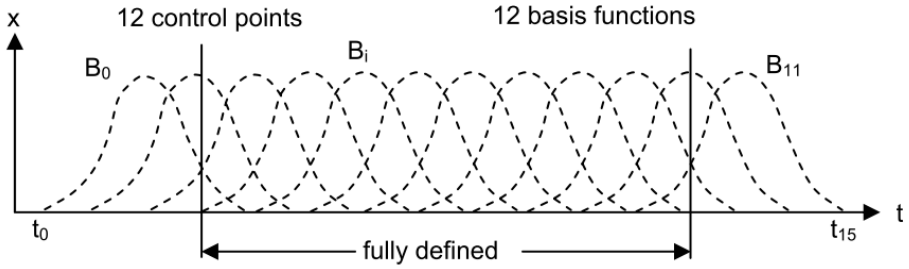

Figure 1.6: Fully defined region of a (uniform) B-spline - an example.

From Figure 1.6, we can make the following observations:

- There are $0, \ldots, m$ ($m + 1$ in total) control points, with $m = 11$.
- There are as many basis functions as the control points, thus m+1 (= 12). Each basis function 'uses' (k+1=) 5 knots.

- The total number of knots is $m + 1 + (k + 1) - 1 = m + 5$, here 16.
- By inspecting the above figure, we can see that that the curve is not fully defined over the first 3 and the last 3 bays because there are fewer than ($k =$) 4 basis functions. As a resultm, a uniform B-Spline curve does not start on the first control point nor does it end at the last control point.

In the next Chapter we are also going to need the *time derivative* of a cubic B-spline in parametric form. First we apply the chain rule (to be more precise q(u) only for this step is written as q(u(t))) :

$$\frac{d}{dt}q(u(t)) = \frac{d}{du}q(u(t))\frac{d}{dt}u(t)$$

Then by using equations 1.19 and 1.20, the time derivative becomes

$$\frac{d}{dt}q(u(t)) = \frac{1}{2(t_{i+1} - t_i)}\begin{bmatrix} u^2 & u & 1 \end{bmatrix}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -4 & 2 & 0 \\ -1 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \end{bmatrix} \tag{1.21}$$

Since we are working with uniform B-splines the distance between two consecutive knots will always remain constant and equal to :

$$h = t_{i+1} - t_i \ , \ \forall i \tag{1.22}$$

1.3.4 Useful B-Splines properties

Some properties of B-Splines are going to be discussed here, since they are of particular interest for trajectory optimization. We are going to focus on the special case of cubic B-splines but similar properties apply to B-Splines of arbitrary order. For abbreviation cubic B-Splines are going to be called *CBS*.

Property 1: CBSs are C^2 continuous.

In case of trajectory optimization this is important because continuity of velocity and acceleration makes the task easier for the joint motors, in terms of the torque that has to be applied.

Property 2: Each control point has only local effects on the resulting CBS.

This is an obvious fact, that arises from the definition of B-Splines. Each basis function only affects a neighbourhood of the final curve. The size of the neighborhood is equal to the order of the spline.

Property 3: Convex Hull property.

For a trajectory described as in 1.16, the inequality

$$\underline{q} \leq q(t) \leq \overline{q}$$

implies that

$$\underline{q} \leq p_i \leq \overline{q} \ , \ i = 0, \ldots, m$$

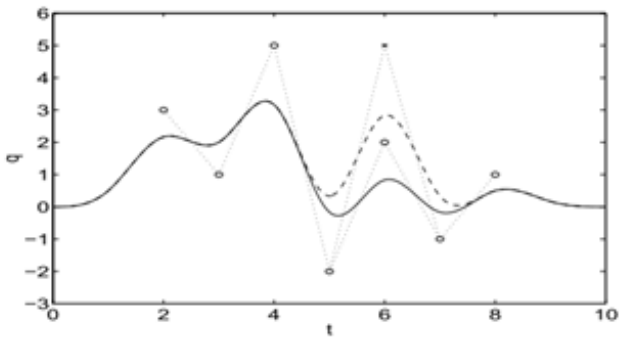

Figure 1.7: Basis functions that affect $q(u)$ for $u \in [0, 1]$

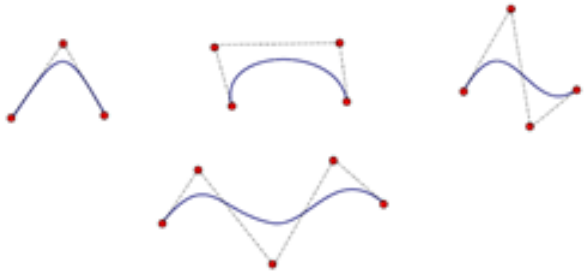

Figure 1.8: Illustration of the convex hull property.

In other words, the B-Spline curve is contained in the convex hull of its control points (also known as control polyline). Some examples are shown in Figure 1.8.

Property 4: The derivative of a B-Spline of order k, is also a B-Spline of order $k-1$.

Proof: Assume a trajectory described by a B-Spline, as in 1.16:

$$q(t, P) = \sum_{j=0}^{m} p_j B_{j,k}(t)$$

It is clear that

$$\frac{d}{dt} q(t, P) = \sum_{j=0}^{m} p_j \frac{d}{dt} B_{j,k}(t)$$

The derivative of each of the Basis functions can be computed as follows:

$$\frac{d}{dt} B_{j,k}(t) = \frac{k}{t_{i+k} - t_i} B_{j,k-1}(t) - \frac{k}{t_{i+k+1} - t_{i+1}} B_{j+1,k-1}(t)$$

Plugging these derivatives back to the curve equation yields:

$$\frac{d}{dt} q(t, P) = \sum_{j=0}^{m-1} h_i B_{j+1,k-1}(t)$$

where

$$h_i = \frac{k}{t_{i+k+1} - t_{i+1}} (p_{i+1} - p_i)$$

Therefore the derivative of a B-Spline curve is another B-Spline curve of degree k-1 on the initial knot vector with a new set of control points $h_0, \ldots, h_{k-1}$.

Chapter 2

Putting it all together

In the previous Chapter, the general optimization problem to solve has been presented and then some tools where discussed, namely, computed-torque control and LQR control for the linear outer loop feedback and then B-Splines in order to describe a manipulator trajectory.
So far the optimization problem is (section 1.1) :

$$\tau^* = argminJ$$

Subject to:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau$$

$$q(0) = q_0, q(t_f) = q_f$$

(additional physical constraints)

The objective function J, could have many different expressions based on what we are trying to optimize. Here the energy efficiency of the state transition $q_0 \rightarrow q_f$ is considered, or the effort that the joint motors have to put in the double pendulum for this state transition to be achieved.

$$J = \frac{1}{2}\int_0^{t_f} \|\tau(t)\|_2^2 \, dt = \frac{1}{2}\int_0^{t_f} \tau(t)^T\tau(t)dt \tag{2.1}$$

Next, we have to introduce the difference between the actual and the desired trajectory of the manipulator. Define the tracking error as before:

$$e(t) = q_d(t) - q(t) \tag{2.2}$$

We assume that the desired trajectory coincides with the actual trajectory initially, thus:

$$q_d(0) = q(0)$$

2.1 Desired joint trajectory description using uniform cubic B-splines

Now we will describe the desired trajectory of each joint as a uniform cubic B-spline that satisfies the given initial and final conditions. First, the desired joint trajectory using the classical B-spline representation will be:

$$q_d(t) = \begin{bmatrix} \theta_{1d}(t) \\ \theta_{2d}(t) \end{bmatrix} = \begin{bmatrix} \sum_{k=0}^{m} p_{1k} B_{k,4}(t) \\ \sum_{k=0}^{m} p_{2k} B_{k,4}(t) \end{bmatrix} \tag{2.3}$$

The control points that produce both joint trajectories are contained in vector P:

$$P = \begin{bmatrix} P_1 \\ P_2 \end{bmatrix} \text{ , } with\ P_i = \begin{bmatrix} p_{i0} & \dots & p_{im} \end{bmatrix}^T \text{ , } i = 1, 2 \tag{2.4}$$

In other words, the vector P_i contains the set of control points that produce the joint trajectory of joint i. For simplicity, we assume that an equal number of control points is used for each joint trajectory (m+1 control points) but this can be easily extended anyway.
Up to this point, we do not have any information about the values of vector P, apart from the fact that the resulting trajectories should satisfy the given initial and final conditions. By intuition, one would expect that the given conditions will impose some constraints on the values that the control points can get in the neighbourhood of $t = 0$ and $t = t_f$ (recall section 1.3.4, Property 2; each control point has only local effects). It is convenient to investigate this constraint by using the parametric formulation of the B-spline trajectory.
By keeping in mind that on initial and final time there will always be a knot point, we can take the value of u=0. In fact we can obtain the same value by taking u = 1, based on the facts that there will be additional control points before initial time and after final time (recall the region that a B-spline is fully defined) and that continuity between curve segments is guaranteed (section 1.3.4, Property 1). Given initial conditions:

$$q_0 = \begin{bmatrix} \theta_{10} & \theta_{20} & \dot{\theta}_{10} & \dot{\theta}_{20} \end{bmatrix}^T$$

From equation 1.20, with u=0, follows that:

$$p_{j,i-2} + 4p_{j,i-1} + p_{j,i} = 6\theta_{j0} \text{ , } j = 1, 2 \tag{2.5}$$

Similarly the angular velocity values on t=0 give the following equation, using 1.21 with u=0:

$$-p_{j,i-2} + p_{j,i} = 2h\dot{\theta}_{j0} \text{ , } j = 1, 2 \tag{2.6}$$

Recall that h is the distance between two consecutive knots (equation 1.22). We can reach to similar equations by requiring the final conditions to be true:

$$q_f = \begin{bmatrix} \theta_{1f} & \theta_{2f} & \dot{\theta}_{1f} & \dot{\theta}_{2f} \end{bmatrix}^T$$

Leads to:

$$p_{j,i-2} + 4p_{j,i-1} + p_{j,i} = 6\theta_{jf} \text{ , } j = 1, 2 \tag{2.7}$$

and

$$-p_{j,i-2} + p_{j,i} = 2h\dot{\theta}_{jf} \text{ , } j = 1, 2 \tag{2.8}$$

A subtle problem on the optimization problem that we want to solve is the fact that the time needed to go from initial to final state (will be referred to as final time, t_f, since $t_0 = 0$) is unknown; therefore it is a variable that we have to optimize as well. This fact also makes the optimization problem more difficult because the constraints imposed on the control points in equations 2.5 - 2.8, become *non-linear*!

2.2 Control Scheme

Now that the desired trajectories have been defined using B-splines, the tracking error of the double pendulum can be written as follows:

$$e(t) = \begin{bmatrix} \sum_{k=0}^{m} p_{1k} B_{k,4}(t) - \theta_1(t) \\ \sum_{k=0}^{m} p_{2k} B_{k,4}(t) - \theta_2(t) \\ \sum_{k=0}^{m} p'_{1k} B_{k,3}(t) - \dot{\theta}_1(t) \\ \sum_{k=0}^{m} p'_{2k} B_{k,3}(t) - \dot{\theta}_2(t) \end{bmatrix} \tag{2.9}$$

Recall from section 1.3.4, Property 4, that the time derivative of a B-spline of order k is also a B-spline of order k-1 having control points that are linearly related to the initial ones.
Next, the computed torque control scheme is going to be applied. As it was shown in section 1.2.2 the torque that we have to apply in order to drive the tracking error to zero is

$$\tau = M(q)(\ddot{q_d} - u) + N(q, \dot{q}) \tag{2.10}$$

On every timestep, in order to calculate the torque that each motor has to produce the following calculations take place:

- Find tracking error,e, using 2.9.
- Calculated u, using LQR:
  $$u = -Ke$$
  where K is only once calculated from equations 1.13, 1.14.
- Using 2.10 the torque that each motor has to apply can be calculated. Since the current state and time is assumed to be known all the elements in 2.10 can be calculated.

The derivation of matrices M,N for a double pendulum can be found in Appendix A. Given the facts that the double pendulum is fully actuated and the initial actual and desired state coincide we can guess that the tracking is not going to be really challenging.
However, by using such approach, we do not have to worry about tuning or calculating any control parameters during the optimization procedure. We can just focus on tuning the trajectory that the pendulum has to follow.
Before proceeding to writing down the final optimization problem, some additional constraints are going to be expressed in this framework.

2.3 Angular Velocity Constraints

Angular velocity constraints make sense practically since the joint motors impose bounds on the angular velocity, the motor shaft can rotate with. Here only the case of the double pendulum is considered but the extension to more complex manipulators is straightforward. Additionally for simplicity the same bounds apply on all joints (also easy to extend).

$$\underline{\dot{\theta}} \leq \theta_i \leq \overline{\dot{\theta}}\ ,\ i = 1, 2 \tag{2.11}$$

B-splines make the integration of such constraints to the optimization problem easy. Recall section 1.3.4, Property 4: The derivative of a B-Spline of order k, is also a B-spline of order k-1. Since the joint desired trajectories are described using cubic B-splines, their time derivatives will be quadratic B-splines. The parametric form of quadratic B-spline is given by equation 2.12.

$$q(u) = \begin{bmatrix} u^2 & u & 1 \end{bmatrix} \frac{1}{2} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \hat{p}_{i-1} \\ \hat{p}_i \\ \hat{p}_{i+1} \end{bmatrix} = \begin{bmatrix} u^2 & u & 1 \end{bmatrix} \frac{1}{2} M_1 \begin{bmatrix} \hat{p}_{i-1} \\ \hat{p}_i \\ \hat{p}_{i+1} \end{bmatrix} \tag{2.12}$$

Additionally if we take the time derivative of cubic B-spline the result is (this was derived earlier in equation 1.21):

$$\frac{d}{dt} q(u(t)) = \frac{1}{2h} \begin{bmatrix} u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -4 & 2 & 0 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \end{bmatrix} = \frac{1}{2h} \begin{bmatrix} u^2 & u & 1 \end{bmatrix} M_2 \begin{bmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \end{bmatrix}$$

From equations 1.21 and 2.12 the following relation between the control points of a cubic B-spline and it's time derivative counterparts holds:

$$\begin{bmatrix} \hat{p}_{i-1} \\ \hat{p}_i \\ \hat{p}_{i+1} \end{bmatrix} = M_1^{-1} M_2 \begin{bmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \end{bmatrix} = \begin{bmatrix} -1/h & 1/h & 0 & 0 \\ 0 & -1/h & 1/h & 0 \\ 0 & 0 & -1/h & 1/h \end{bmatrix} \begin{bmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \end{bmatrix} \tag{2.13}$$

Next, property 3 from section 1.3.4 is going to be used; the convex hull property. The inequality constraints 2.11 can be directly related to the control points $\hat{p}_i$:

$$\underline{\dot{\theta}} \leq \hat{p}_i \leq \overline{\dot{\theta}} \,,\; i = 0, \ldots, m \tag{2.14}$$

Finally, using equations 2.13 and 2.14 the angular velocity constraints can be translated to constraints on the control points:

$$\begin{bmatrix} -1/h & 1/h & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & -1/h & 1/h & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & -1/h & 1/h & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 1/h & -1/h & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1/h & -1/h & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 & 1/h & -1/h \end{bmatrix} \begin{bmatrix} p_{i0} \\ p_{i1} \\ \vdots \\ p_{im} \\ p_{i0} \\ p_{i1} \\ \vdots \\ p_{im} \end{bmatrix} \leq \begin{bmatrix} \overline{\dot{\theta}} \\ \overline{\dot{\theta}} \\ \vdots \\ \overline{\dot{\theta}} \\ -\underline{\dot{\theta}} \\ -\underline{\dot{\theta}} \\ \vdots \\ -\underline{\dot{\theta}} \end{bmatrix}, \; i = 1, 2 \tag{2.15}$$

It should be noted again that the constraints in 2.15 are non-linear because the final time is an optimization variable.[1]

2.4 Torque Constraints

Under this framework torque constraints cannot be directly related to the optimization parameters (control points, final time). The methodology proposed in [13], [14] was used instead; apply torque constraints as *soft constraints*. The idea is to introduce the deviation outside the allowed bounds of torque to the objective function that we want to minimize. Assume that certain bounds are given for the torques of the joint motors:

$$\underline{\tau} \leq \tau(t) \leq \overline{\tau} \tag{2.16}$$

First the following functions are defined for each joint (i):

$$\begin{aligned} \phi_{i1}(t) =& max\left(\underline{\tau} - \tau_i(t), 0\right) \\ \phi_{i2}(t) =& max\left(\tau_i(t) - \overline{\tau}, 0\right) \end{aligned} \tag{2.17}$$

Then a new term, J_τ is introduced on the objective function:

$$\begin{aligned} J_\tau =& \frac{1}{2}\int_0^{t_f} \|\phi(t)\|_2^2 \, dt \\ =& \frac{1}{2}\int_0^{t_f} \left\| \begin{bmatrix} \vdots \\ \phi_{i1}(t) \\ \phi_{i2}(t) \\ \vdots \end{bmatrix} \right\|_2^2 dt \end{aligned} \tag{2.18}$$

2.5 The Optimization Problem Rewritten

The initial form of the problem was stated in the beginning of the chapter. The ultimate goal of finding the torques that minimize the effort is going to determined indirectly,

[1] Since the number of control points is fixed (therefore the number of knots is also fixed) for every value of t_f the knot points (t_i) are redistributed so that they evenly cover the t-axis, thus :

$$h = t_{i+1} - t_i$$

takes a different value.

by fixing a control strategy and tuning the desired trajectory instead. The optimization variables under this approach become the control points of the cubic B-splines that describe the joints' trajectories and the final time. By putting all the pieces of the previous sections together we can see how this is possible.
We start from the initial definition of the objective function (equation 2.1):

$$\begin{aligned}
J =& \frac{1}{2}\int_0^{t_f} \|\tau(t)\|_2^2\, dt \\
=& \frac{1}{2}\int_0^{t_f} \|M(q(t))(\ddot{q}_d(t) - u(t)) + N(q(t),\dot{q}(t))\|_2^2\, dt \ \textit{equation } 2.10 \\
=& \frac{1}{2}\int_0^{t_f} \|M(q(t))(\ddot{q}_d(t) + Ke(t)) + N(q(t),\dot{q}(t))\|_2^2\, dt \\
=& \frac{1}{2}\int_0^{t_f} \left\| M(q(t))(\ddot{q}_d(t) + K \begin{bmatrix} q(t) - q_d(t) \\ \dot{q}(t) - \dot{q}_d(t) \end{bmatrix}) + N(q(t),\dot{q}(t)) \right\|_2^2 dt
\end{aligned}$$

where

$$q_d(t) = \begin{bmatrix} \sum_{k=0}^{m} p_{1k} B_{k,4}(t) \\ \sum_{k=0}^{m} p_{2k} B_{k,4}(t) \end{bmatrix}$$

The optimization variables can all be stored inside a vector P :

$$P = \begin{bmatrix} P_1 & P_2 & t_f \end{bmatrix} \tag{2.19}$$

where P_1 is the vector of the control points of the desired trajectory of 1^{st} joint, P_2 is the vector of the control points of the desired trajectory of 2^{nd} joint and t_f is the final time. Now we can write the optimization problem on its final form:

Unconstrained Optimization Problem

$$\min_P \frac{1}{2}\int_0^{t_f} \left\| M(q(t))(\ddot{q}_d(t) + K \begin{bmatrix} q(t) - q_d(t) \\ \dot{q}(t) - \dot{q}_d(t) \end{bmatrix}) + N(q(t),\dot{q}(t)) \right\|_2^2 dt$$

Subject to:

$$\begin{bmatrix} 1 & 4 & 1 & 0 & \dots & 0 & 0 \\ -1 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & 1 & 4 & 1 & 0 \\ 0 & \dots & 0 & -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_{10} \\ \vdots \\ p_{1m} \\ p_{20} \\ \vdots \\ p_{2m} \\ t_f \end{bmatrix} = \begin{bmatrix} 6\theta_{1f} \\ 2h\dot{\theta}_{1f} \\ 6\theta_{2f} \\ 2h\dot{\theta}_{2f} \end{bmatrix}$$

*where $h = h(t_f)$

By introducing additionally the angular velocity and torque constraints the problem is written as follows:

Constrained Optimization Problem

(Torque and angular velocity constraints)

$$\min_{P} \frac{1}{2} \int_0^{t_f} \left\| M(q(t))(\ddot{q}_d(t) + K \begin{bmatrix} q(t) - q_d(t) \\ \dot{q}(t) - \dot{q}_d(t) \end{bmatrix}) + N(q(t), \dot{q}(t)) \right\|_2^2 dt + W \frac{1}{2} \int_0^{t_f} \|\phi(t)\|_2^2 \, dt$$

Subject to:

$$\begin{bmatrix} 1 & 4 & 1 & 0 & \dots & 0 & 0 \\ -1 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & 1 & 4 & 1 & 0 \\ 0 & \dots & 0 & -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_{10} \\ \vdots \\ p_{1m} \\ p_{20} \\ \vdots \\ p_{2m} \\ t_f \end{bmatrix} = \begin{bmatrix} 6\theta_{1f} \\ 2h\dot{\theta}_{1f} \\ 6\theta_{2f} \\ 2h\dot{\theta}_{2f} \end{bmatrix}$$

$$\begin{bmatrix} -1/h & 1/h & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & -1/h & 1/h & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & -1/h & 1/h & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 1/h & -1/h & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1/h & -1/h & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 & 1/h & -1/h \end{bmatrix} \begin{bmatrix} p_{i0} \\ p_{i1} \\ \vdots \\ p_{im} \\ p_{i0} \\ p_{i1} \\ \vdots \\ p_{im} \end{bmatrix} \leq \begin{bmatrix} \bar{\dot{\theta}} \\ \bar{\dot{\theta}} \\ \vdots \\ \bar{\dot{\theta}} \\ -\underline{\dot{\theta}} \\ -\underline{\dot{\theta}} \\ \vdots \\ -\underline{\dot{\theta}} \end{bmatrix}, \; i = 1, 2$$

*where $h = h(t_f)$

2.6 Solving the optimization problem

The solution of the optimization problem is going to take place in $Matlab^{TM}$ environment. Since the parameter space is relatively small, some medium scale algorithm can be used. The difference between large and medium scale algorithms lies on the need of storing and operating on full matrices, i.e. matrices that each element is explicitly stored. On large parameter space due to large memory consumption of full matrix operations, large scale algorithms are preferred; in such case the matrices are stored as sparse matrices and the internal algorithms are maintaining this sparsity. By testing different medium and large scale algorithms on this problem the selection of medium scale algorithm made sense since the difference in speed was very small and also the performance of medium scale algorithms was significantly better in terms of the quality of the local minimum and the convergence behavior.

In terms of performance and quality of the local minimum the *Sequential Quadratic Programming (SQP)* algorithm appeared to be the best candidate. SQP methods represent the state of the art in non-linear programming methods. The main idea behind this method is the formulation of a Quadratic Programming subproblem based on a quadratic approximation of the Lagrangian function. The details of this procedure are going to be skipped

here.
Assume the general non-linear optimization problem with non-linear constraints:

$$\min_x f(x)$$

Subject to:

$$g(x) \leq 0$$

Then the Lagrangian function is defined as :

$$L(x, \lambda) = f(x) + \sum_{i=1}^{n} \lambda_i g_i(x)$$

In order to obtain the QP approximating subproblem the quadratic approximations of the objective function and non-linear constraints are going to be calculated, defining as x_k the values of the variable vector x at iteration k.

- Objective function quadratic approximation:

$$f(x) \approx f(x_k) + \nabla f(x_k)(x - x_k) + \frac{1}{2}(x - x_k)^T H f(x_k)(x - x_k)$$

  Here we just did a second order Taylor approximation of the objective function and used the H symbol for the Hessian *operator* of the objective function which is $H = \nabla^2$.

- Constraint functions g local affine approximation:

$$g(x) \approx g(x_k) + \nabla g(x_k)(x - x_k)$$

In the literature usually the following definitions are used:

$$d(x) = x - x_k$$

and

$$B_k = H f(x_k)$$

Then the QP subproblem is written as follows:

$$\min_x \nabla f(x_k) d(x) + \tfrac{1}{2} d(x)^T B_k d(x)$$

Subject to:

$$g(x_k) + \nabla g(x_k)^T d(x) \leq 0$$

The QP subproblem can be solved using any QP algorithm. Then given that d(x) has been calculated the optimization variable, x, is updated:

$$x_{k+1} = x_k + a_k * d(x)$$

The term a_k is the step length; without getting into the details on each iteration the step size is calculated using a line search method such that the step is as big as possible without leading to violations of any constraint.
The concepts of linear and non-linear programming that were mentioned here rely on some very rich literature. Some sample starting points are [7], [6] and [8](scribe notes). Lastly it should be noted that even though such analysis may go beyond the scope of this project, by knowing more explicitly how this optimization procedure works can lead to ideas for further improving the optimization procedure (Chapter 4).

Chapter 3

Simulation

3.1 Introduction

On Chapter 2 the optimization problems have been derived in order to produce minimum effort motions between given initial and final conditions with or without torque and angular velocity constraints. Here some numerical results of the proposed method are going to be illustrated. Some fixed model parameters for the double pendulum are considered throughout the entire Chapter (Table 3.1).[1]

Parameter	**Value**
$m_1,\ m_2$	$0.5 \quad kg$
$l_1,\ l_2$	$0.5 \quad m$
$r_1,\ r_2$	$0.25 \quad m$
$I_{z1},\ I_{z2}$	$0.01 \quad kg\, m^2$

Table 3.1: The double pendulum model parameters that were used. For explanation of the symbols refer to Figure 1.1.

As it was mentioned in Section 2.6, the $Matlab^{TM}$ environment is used for solving the problem.

3.2 'Shoulder' movement by 90 degrees - A benchmark task

As a benchmark, the task of moving the shoulder joint (joint 1) from 0 to 90 degrees is considered, having zero velocities at initial and final position (Figure B.2). This task was chosen because intuitively the quality of the result can be roughly estimated. Given that the gravitational field has the direction shown in Figure B.2, it makes sense to avoid going directly to the final position and perform some swinging motion instead.
The optimization algorithm is gradient based thus some initial guesses of the optimization variables has to be given. Instead of giving random initial guesses a simple polynomial interpolation is used as an initial guess. The main motivation behind this choice is because

[1] cylindrical links are assumed, therefore:

$$I_{zi} = \frac{2}{3} m_i l_i^2$$

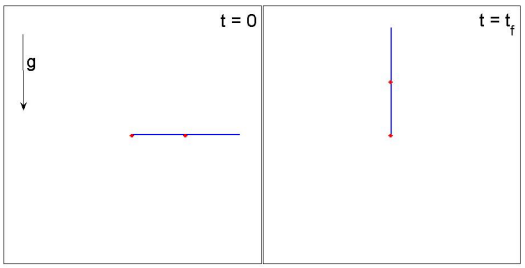

Figure 3.1: The problem to solve.

comparison of the performance of the algorithm for different parameter sets is easier, compared to the case of totally random initial values. In order to achieve that the initial values of the control points for each trajectory are tuned such that the resulting trajectory coincides with the cubic polynomial trajectory that satisfies the initial and final velocity and angular velocity conditions (Figure 3.2).The initial guess of the final time is kept fixed and it is equal to *2 sec*.

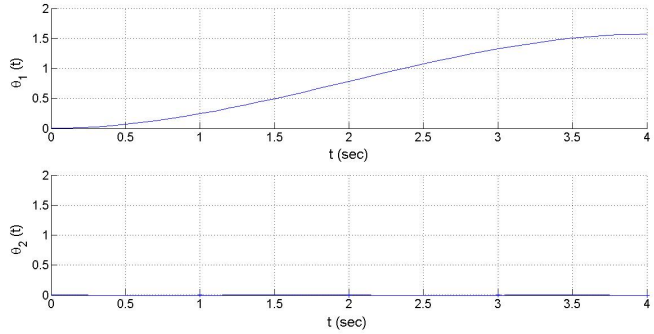

Figure 3.2: The initial guess of each joint trajectory (cubic polynomial interpolation between initial and desired final condition satisfying the initial and desired final angular velocity values).

Apart from cases of low dimensionality (5 to 8 control points per joint trajectory) and mostly on unconstrained case, the local minimum values can vary significantly on each run. Thus, no strong statement can be made about the optimal number of control points. However a clear drop on the performance can be generally observed on the extreme cases. Very small number of control points (5-6) is somewhat limiting in terms of how much the trajectory can be tuned and therefore some swinging motions cannot be exploited. On the other hand a large number of control points makes the algorithm reach poor local minima more frequently. Additionally a large number of control points increases the computational cost of the algorithm. In this example, 7 to 9 control points seeemed to give a good trade-off between computational cost and quality of the local minimum that the algorithm can find.
In Figures 3.3, 3.4 some results are shown for the shoulder movement task. As a baseline for comparison the objective function value is plotted assuming cubic polynomial interpolation between the initial and final conditions, for varying values of final time, t_f. Moreover, in order to make the comparison more fair it makes sense to use as a baseline the minimum value of J that can be achieved using polynomial interpolation.
Comments on the Figures 3.3, 3.4:

- The effectiveness of the algorithm can be observed in both constrained and uncon-

strained cases.

- Various different types of swinging motions have been found giving values of J^* approximately between 3 and 5. This also gives a feeling about the big number of local minima of the objective function. [2]
- Interestingly, by imposing harder constraints on the problem does not necessarily lead to worse values of J^*. The same statement can be made about the number of control points. The most distinctive example for this claim is point (2) in the constrained case, having constraints on both torques and angular velocity. In this case the best value of J^* was obtained, compared to other ones, including the unconstrained case. Also, notice that this value is also better than the value of J^* having the same constraints and one additional control point (m=8 instead of m=7).
- Due to the big number of local minimas the algorithm seems to perform sometimes better under constraints. This observation is sometimes true is non-smooth non-linear optimization problems, since by imposing constraints, the search space becomes more limited and this can leave out of the acceptable region local minimas on which otherwise the algorithm could have converged to. This was also observed during some runs of the algorithm under constraints; there were significantly more fluctuations on the value of the objective function over the number of iterations (in the unconstrained case usually the objective function converges to some minimum without significant fluctuations). An additional reason for this behavior is the numerical approximation of the gradient and the Hessian of the objective function(non-smooth regions can lead to poor approximation of these matrices). However, these fluctuations of the objective function value make the algorithm to converge significantly slower.
- It is important to notice that the solutions of the optimization algorithm not only provide trajectories with smaller values of J, but also by imposing constraints, *feasible* solutions are obtained. Under polynomial interpolation of the initial and final conditions the maximum torques for t_f giving best value of J are around 9 Nm, while it can be seen that smaller values of J can be obtained (up to 80% smaller) by keeping the torques whithin the bounds of ± 2 Nm and also limiting the angular velocity acceptable values.

[2] Since the stop motion graph cannot fully describe the trajectory, the interested reader could refer to the deliverable's attached material where the corresponding videos for each result can be found.

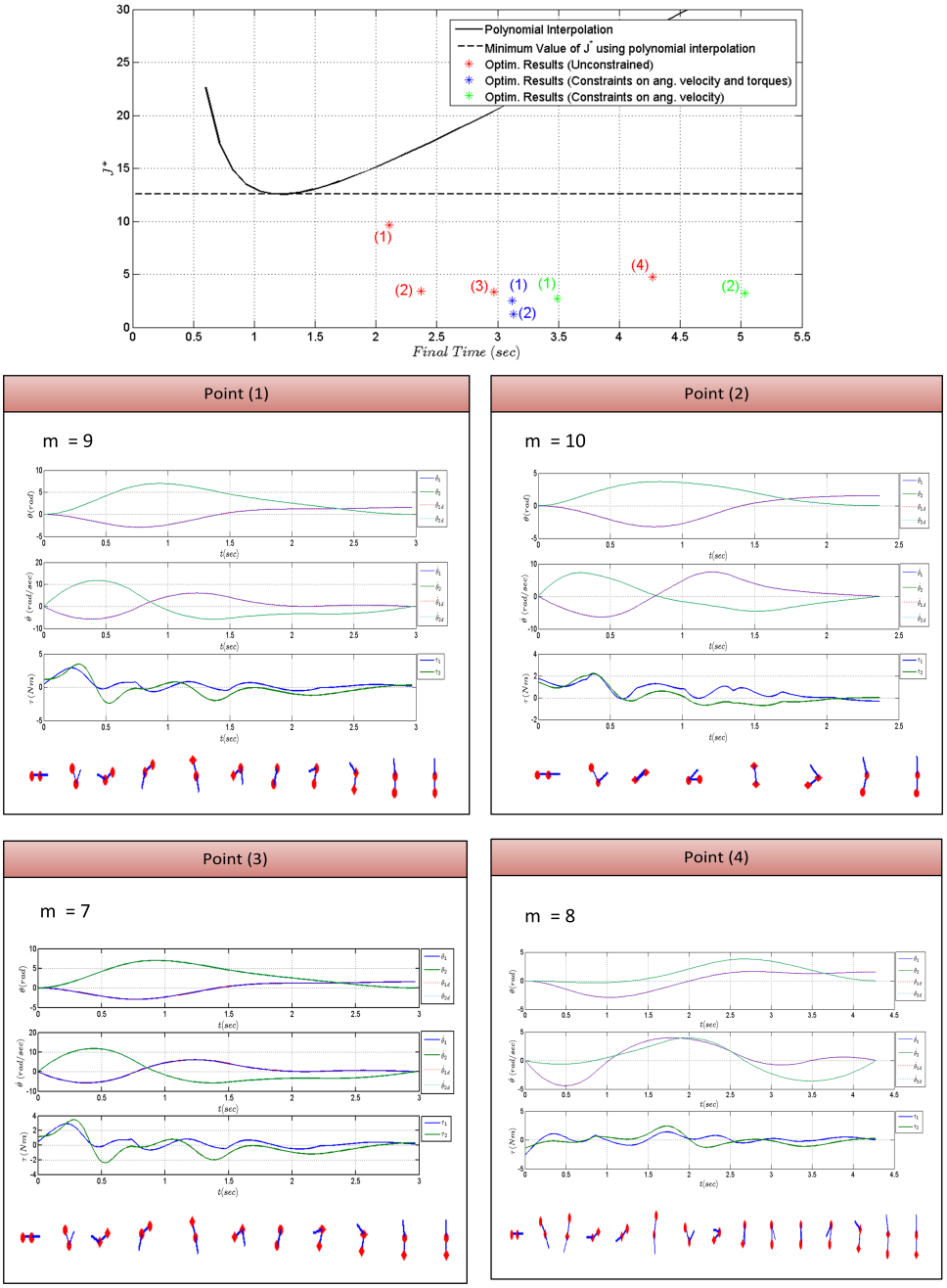

Figure 3.3: Various optimization results for the shoulder movement task compared to the values of J that can be obtained using cubic polynomial interpolation between the initial and final conditions.

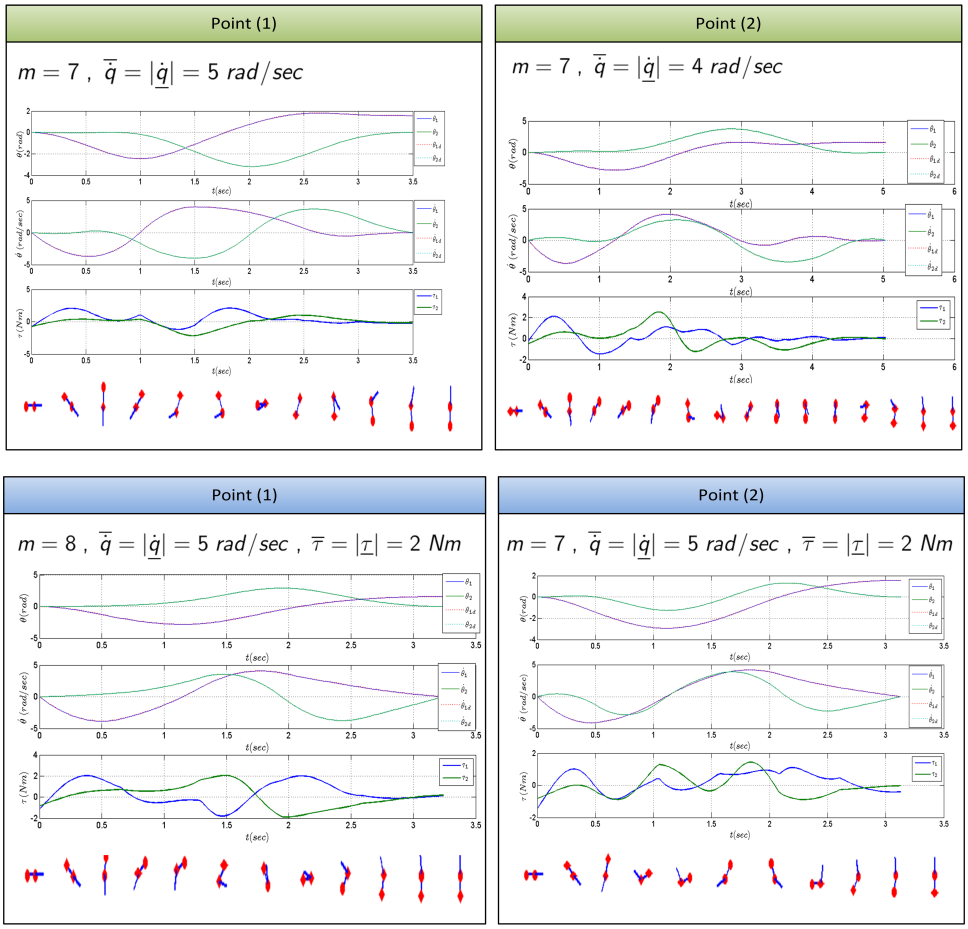

Figure 3.4: Figure 3.3 continued.

3.3 Throwing motion

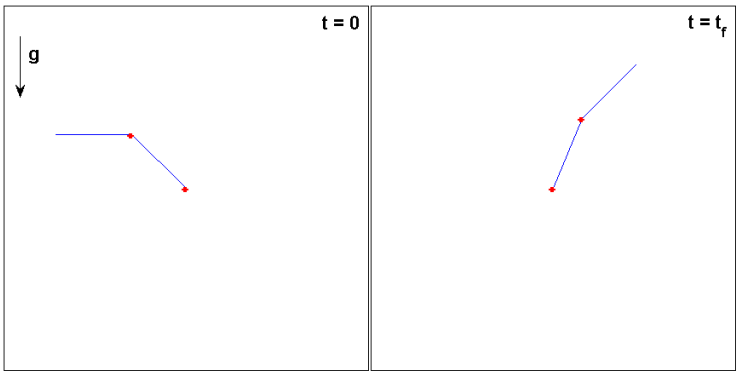

Figure 3.5: The Throwing task.

The exact values for the desired initial and final conditions are:

$$q_{d0} = q_0 = \begin{bmatrix} \frac{3\pi}{4} \\ 0 \\ 0 \\ 0 \end{bmatrix} \ , \ q_{df} = \begin{bmatrix} \frac{3\pi}{8} \\ -\frac{pi}{8} \\ -3 \\ -6 \end{bmatrix}$$

This task was chosen as an example of a throwing-like motion. Again the main purpose is to investigate whether swing-like motions can give a better result in terms of the effort objective function. In the following chapter extensions of this task will be discussed.
In Figure 3.6 some optimization results are shown, again compared to the cubic interpolation case.
Comments on the results:

- The different types of local minima in this problem are reduced. Especially in case of unconstrained optimization, there is one dominating minimum which can be seen in the Figure. The difference of the value of the objective function in the unconstrained case mostly relates to slightly different timing of the swings but the motion is quite similar.
- Again in this case better values of J were found by adding constraints to the problem. Again the convergence time is significantly increased, especially when torque constraints exist and significantly more fluctuations of the value of J over the iterations is observed.
- In this example, the best value of J was obtained when only torque constraints are imposed and the value of J we obtain is again around 80 % smaller, compared to the best J we can obtain using polynomial interpolation. Additionally the torques are bounded within $\pm 3.5\ Nm$ compared to about $14\ Nm$ that is roughly the maximum torque demand in the later case.

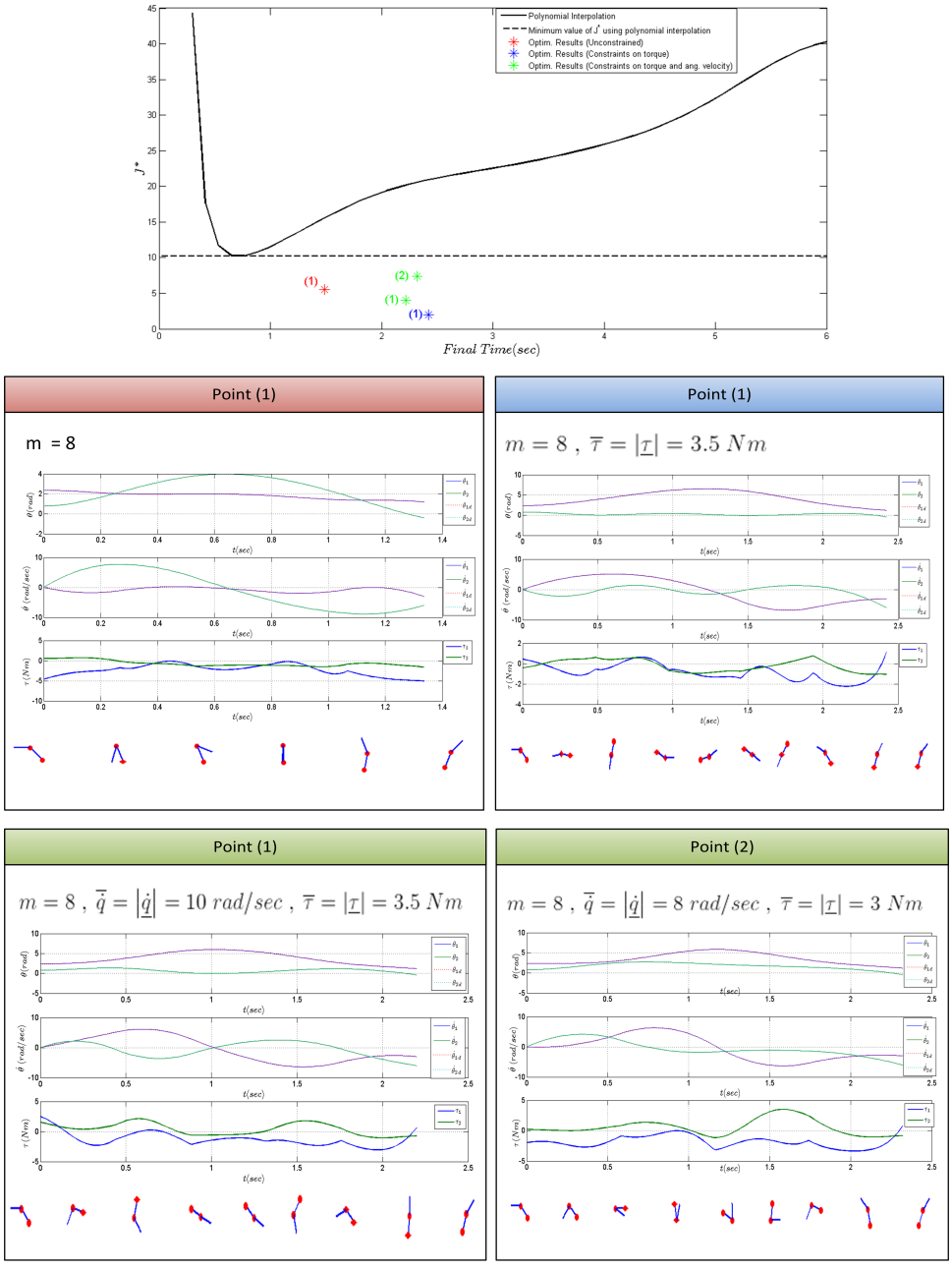

Figure 3.6: Various optimization results for the throwing task compared to the values of J that can be obtained using cubic polynomial interpolation between the initial and final conditions.

Chapter 4

Conclusion

The main ingredients of the approach that was described are the fixed control strategy and the transformation of the optimal trajectory problem into a discrete optimization problem (we are optimizing the control points and the final time) in order to minimize the effort of the joint motors for achieving a task. Torque and angular velocity constraints were taken into account and actually in both cases this helped in finding better local minima (which of course satisfy the imposed constraints as well) by limiting the search space.
The advantages of this approach are:

- Modularity. Different objective functions and physical models can be added easily.
- The obtained results are quite easy to apply since the desired trajectory can be reproduced by transferring a small number of parameters (for example if we use 10 control points per joint trajectory then the controller of the double pendulum needs $20 + 1$(the final time) parameters in order to reproduce the desired trajectory).
- The produced trajectories are guaranteed to be smooth which leads to smooth transitions in torque demands, something that is desirable for reducing the wearing of the motors.
- Swinging motions can be discovered that take advantage of the system's inherent dynamics and stored energy.

The disadvantages of this approach are:

- The optimization problem that has to be solved is non-smooth and non-convex. As a result, it can get stuck to poor local minimas. This makes the methodology not a 'one button solution', since in order to obtain some good results experimentation is needed for the initial guesses and the number of control points.
- It is an off-line method. This makes it practical only for experimentation and repetitive tasks.

There are many things that can improve this approach:

- The optimization performance is expected to be significantly increased if analytical expressions of the gradient and (even better the Hessian also) of the objective function and the constraints are available to the solver. Although this is not a trivial task, some guidelines exist ([12]), in order to systematically obtain the gradient by using the Newton-Euler formulation of the system's dynamics and the properties of matrix exponentials.
- The effect of damping can be explored on the resulting optimal trajectories.
- The fact that the final time is not known makes the boundary condition and angular velocity contraints non-linear. Also this contributes a lot to the 'non-smoothness' of the problem. A different heuristic approach for finding the final time outside the optimization loop, proposed in [16] would make sense to try and see if performance improves.
- Torque derivative constraints would be a good addition for making sure that the resulting trajectories are truly feasible for given hardware.

- For optimizing throwing tasks:
  - A mass can be added at the end effector position, resembling the object that is going to be thrown. The effect of the mass on the resulting optimal trajectories can then be explored (as in [25])
  - The advantage of minimizing the effort can be exploited to maximize the throwing distance for example. This can be achieved by properly changing the objective function (a similar application can be found in [13]).
  - The final conditions could also become subject to optimization. For example an alternative scenario would be to find the optimal trajectory and final posture/angular velocities in order to achieve a certain velocity (vector) on the end-effector, or even better in order to have the maximum velocity vector norm in some specific direction.

Appendix A

Equations of motion of a double pendulum

The closed-form equations of motion of a double pendulum are going to be derived using Lagrange's equations of motion ([3]). Each link of the pendulum is treated as a rigid body, defined by its mass and the position of the center of mass with respect to its corresponding joint; for example r_1 is the distance of the center of mass of link 1 with respect to joint 1 (Figure A.1).
Our goal, in order to use the Lagrange's equations of motion, is to derive the formulas of the Kinetic and Potential energy of the system. For a rigid body

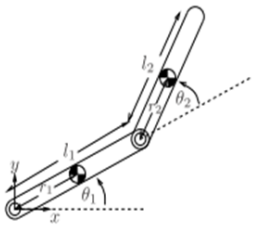

Figure A.1: The double pendulum model

First the Cartesian coordinates (x_i, y_i) of each center of mass are calculated:

$$\begin{aligned} x_1 &= r_1 cos\theta_1 \ \ , \ \ x_2 = l_1 cos\theta_1 + r_2 cos(\theta_1 + \theta_2) \\ y_1 &= r_1 sin\theta_1 \ \ , \ \ y_2 = l_1 sin\theta_1 + r_2 sin(\theta_1 + \theta_2) \end{aligned} \tag{A.1}$$

Then by taking the time derivative of the equations in A.1 we can determine the corresponding velocities of each center of mass with respect the the Cartesian coordinate system:

$$\begin{aligned} \dot{x}_1 &= -r_1 sin\theta_1 \dot{\theta}_1 \ \ , \ \ \dot{x}_2 = -(l_1 sin\theta_1 + r_2 sin(\theta_1 + \theta_2)\dot{\theta}_1 - r_2 sin(\theta_1 + \theta_2)\dot{\theta}_2 \\ \dot{y}_1 &= r_1 cos\theta_1 \dot{\theta}_1 \ \ , \ \ \dot{y}_2 = (l_1 cos\theta_1 + r_2 cos(\theta_1 + \theta_2)\dot{\theta}_1 + r_2 cos(\theta_1 + \theta_2)\dot{\theta}_2 \end{aligned} \tag{A.2}$$

Now the Kinetic Energy (T) of the system can be calculated using [1] :

$$T = \frac{1}{2} m_1 \left\| u_{r1} \right\|_2^2 + \frac{1}{2} I_{z1} \dot{\theta}_1^2 + \frac{1}{2} m_2 \left\| u_{r2} \right\|_2^2 + \frac{1}{2} I_{z2} (\dot{\theta}_1^2 + \dot{\theta}_2^2) \tag{A.3}$$

where

$$u_{ri} = \begin{bmatrix} \dot{x}_i & \dot{y}_i \end{bmatrix}^T$$

[1] For proof about this formula for calculating the kinetic energy of rigid bodies, refer to [3].

is the translational velocity of each center of mass with respect to the Cartesian coordinate system. Each link is modeled as a homogeneous rectangular bar. Its moment of inertia tensor [2] is:

$$I_i = \begin{bmatrix} I_{xi} & 0 & 0 \\ 0 & I_{yi} & 0 \\ 0 & 0 & I_{zi} \end{bmatrix}$$

relative to the frame attached at the center of mass and aligned with the principle axis of the bar.
The Kinetic energy, in A.3 becomes:

$$T = \frac{1}{2} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}^T \begin{bmatrix} \alpha + 2\beta cos(\theta_2) & \delta + \beta cos(\theta_2) \\ \delta + \beta cos(\theta_2) & \delta \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{A.4}$$

where

$$\begin{aligned} \alpha =& I_{z1} + I_{z2} + m_1 r_1^2 + m_2(l_1^2 + r_2^2) \\ \beta =& m_2 l_1 r_2 \\ \delta =& I_{z2} + m_2 r_2^2 \end{aligned}$$

The Potential energy (V) of the system is:

$$\begin{aligned} V =& m_1 g y_1 + m_2 g y_2 \\ =& m_1 g r_1 sin\theta_1 + m_2 g(l_1 sin\theta_1 + r_2 sin(\theta_1 + \theta_2)) \end{aligned} \tag{A.5}$$

The Lagrange's equations of motion are given by:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_i}\right) - \frac{\partial L}{\partial \theta_i} = Y_i$$
$$L = T - V$$

Where Y_i equals to the generalized forces applied on each joint. Since the joints of the double pendulum are rotational the generalized forces refer to torques. Additionally a damping term can be applied (b_i the damping coefficient of joint i):

$$Y_i = \tau_i - b_i \dot{\theta}_i$$

[2] For a rigid body the inertia tensor $I \in R^3$ is defined by:

$$I_i = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$$

with $I^T = I$ (symmetric). Additionally, given that the volume of the rigid body is V and $\rho(r)$ the material density in distance r from the body frame the terms of I are defined by:

$$I_{xx} = \int_V \rho(r)(y^2 + z^2) dxdydz$$
$$I_{xy} = \int_V \rho(r)(xy) dxdydz$$

similarly the rest of the terms can be defined. x,y,z refer to the corresponding coordinates of each 'particle' of the rigid body with respect to the body frame (usually the body frame is attached to the center of mass of the body).

Using equations A.4, A.5 to substitute the energy terms in the Lagrange's equations of motion, we can obtain the equations of motion of the double pendulum. For abbreviation $c_i = cos\theta_i$, $s_i = sin\theta_i$ and $c_ij = cos(\theta_i + \theta_j)$:

$$\begin{bmatrix} \alpha + 2\beta c_2 & \delta + \beta c_2 \\ \delta + \beta c_2 & \delta \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} + \begin{bmatrix} -\beta s_2 \dot{\theta}_2 + b_1 & -\beta s_2(\dot{\theta}_1 + \dot{\theta}_2) \\ \beta s_2 \dot{\theta}_1 & b_2 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} + \\ + \begin{bmatrix} m_1 g r_1 c_1 + m_2 g(l_1 c_1 + r_2 c_1 2) \\ m_2 g r_2 c_1 2 \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \tag{A.6}$$

Appendix B

An overview of trajectory optimization methods

B.1 Introduction

The problem of 'planning' (which is a broader view, that includes the notion of trajectory optimization) is nicely summarized in [17]. The task of planning, trying to be as general as possible, can be summarized by the following basic ingredients:

- **State**. All planning problems involve a state space that captures all the possible states (or situations) that could arise. Usually the size of the state space is too large to be explicitly represented (and exhaustively searched).
- **Time**. All planning algorithms involve a sequence of decisions that must be applied over time. Time can be either explicitly (for example try to reach a state as quick as possible) or implicitly modeled (by reflecting the fact that certain actions must follow in succession). Also time can be discrete or continuous.
- **Actions**. The goal of planning is to generate actions. Somewhere in the planning formulation, it must be specified how the state changes when actions are applied.
- **Initial and goal states**. A planning problem usually involves starting in some initial state and trying to arrive at a specified goal state (which might not be unique). The actions are selected so that this transition will actually happen.
- **A criterion**. This encodes the desired outcome of a plan in terms of the state and actions that are executed. There are two different kinds of planning concerns based on the type of criterion:
  - **Feasibility**. Find a plan that causes arrival at a goal state, regardless of its efficiency.
  - **Optimality**. Find a plan that causes arrival at a goal state and additionally some specified performance is optimized.

  In many cases finding feasible solutions for a planning problem is already a challenging problem without adding any notion of optimality.
- **A plan**. A plan could be a simple sequence of actions to be taken (in control theory this is called 'open-loop') or something more complicated like a function of the current states of the system (feedback plan). It could also be the case that the states cannot be actually measured and the appropriate action must be determined from whatever information is available up to current time. In such cases the actions of a plan are conditioned (probabilistic approach).

There is a clear distinction between trajectory (or path) planning and trajectory optimization. On one hand the goal of planning is to find the set of actions so that the goal state is going to be reached, by also taking feasibility into account. On the other hand, trajectory

optimization pushes the problem one step further and aims to find some optimal set of actions in order to minimize a prescribed objective (known as *objective*, *cost* or *fitness* function) and satisfying the planning requirements at the same time.
The aim of this overview, as the title suggests, is the field of trajectory optimization. A review can be found in [18]; in this paper many references can be found about different problems concerning minimum time, minimum energy and obstacle avoidance optimal trajectory generation approaches, giving also focus on global optimization techniques, usually related to genetic algorithms. Here the different approaches on trajectory optimization are going to be discussed from a different point of view, putting this problem under the optimal control framework. No specific types of objective functions and constraints are considered.
In the following sections each different family of methods is going to be discussed and some references are going to be given as starting points for further research on the topic. Main focus is going to be given to Direct methods which seem to be the most preferable way for tackling with this problem (so far). A summary of the approaches that will be discussed is given in Figure B.1.
We consider the general optimal control problem:

$$J = \underset{x(\cdot),u(\cdot),T}{minimize} \quad \int_0^T L(x(t),u(t))dt + E(x(T))$$

Subject to:

$x(0) - x_0 = 0,$		(fixed initial value)
$\dot{x}(t) - f(x(t),u(t)) = 0,$	$t \in [0,T]$	(ODE model=equations of motion)
$h(x(t),u(t)) \geq 0,$	$t \in [0,T]$	(constraints)
$r(x(T)) = 0,$		(terminal constraints)

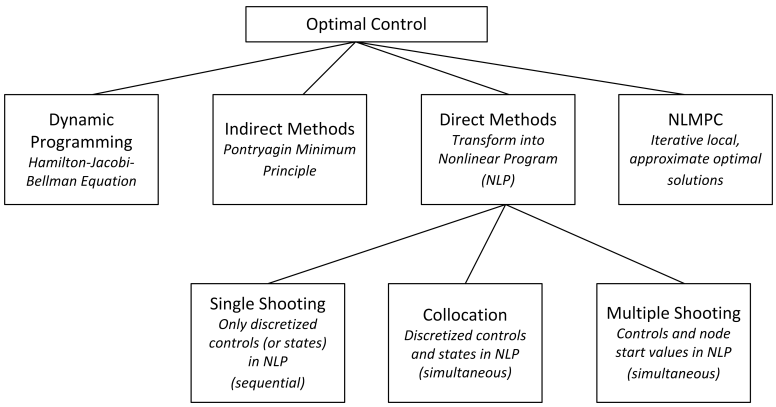

Figure B.1: Trajectory optimization methods.

B.2 Dynamic Programming

The main ingredient of this approach([19]) is Bellman's *principle of optimality*:
An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.
We proceed by introducing the optimal-cost-to-go function on a time interval $[\bar{t}, T]$:

$$J(\overline{x},t) := \min_{x,u} \int_{\bar{t}}^{T} L(x,u)dt + E(x(T)) \ , \ \overline{x} = x(\bar{t})$$

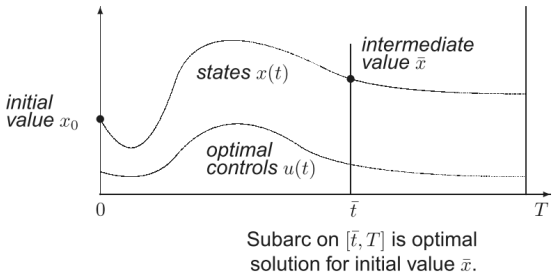

Figure B.2: Applying the principle of optimality.

Consider a grid on the time scale: $0 = t_0 < t_1 < \ldots < t_N = T$. Then we can use the principle of optimality on time intervals $[t_k, t_{k+1}]$ to calculate the optimal-cost-to-go function for that interval:

$$J(x_k, t_k) := \min_{x,u} \int_{t_k}^{t_{k+1}} L(x,u)dt + J(x(t_{k+1}), t_{k+1}) \quad , \quad x(t_k) = x_k \tag{B.1}$$

In other words, equation B.1 states that the optimal J at time t_k will be (if we consider that we move backwards) the optimal J at time t_{k+1} plus the minizing term for the cost of transition from state x_{k+1} to x_k. If this backwards recursion is applied, starting from t_N and moving backwards until t_0 and additionally the time steps are 'infinitely small' we obtain the *Hamilton-Jacobi-Bellman (HJB) equation*:

$$-\frac{\partial J}{\partial t}(x,t) = \min_u \left(L(x,u) + \frac{\partial J}{\partial x}(x,t) f(x,u) \right) \quad , \quad \textit{Subject to } h(x,u) \geq 0 \tag{B.2}$$

with boundary condition:

$$J(x,T) = E(x) \tag{B.3}$$

The optimal J has to be the solution of B.2, thus the optimal control inputs are obtained by solving this partial differential equation (in continuous time) backwards for $t \in [0,T]$, starting at the end of the horizon (equation B.3). In discrete time the fore mentioned dynamic programming backward recursion has to be applied instead.
The strong points of the dynamic programming/HJB approach are:

- The whole state space is searched, thus the global minimum is found.
- We can obtain analytic solutions to some problems

However the main weakness is that solving the HJB equation (or applying dynamic programming in discrete time) can be very challenging especially in high dimensional state space. This is often called in the literature the curse of dimensionality.

B.3 Indirect Methods

The motivation behind this group of methods starts by observing the HJB equation. We obtain the optimal controls by minimizing B.2. Notice that the optimal solution depends only on the derivative $\frac{\partial J}{\partial x}(x,t)$ and not on J itself. Here only the main idea is going to presented (for a more detailed approach refer to [19]) so for now assume that no inequality constraints exist. By introducing the adjoint variables:

$$\lambda(t) := \frac{\partial J}{\partial x}(x(t),t)^T$$

we can obtain the optimal controls from *Pontryagin's Minimum Principle*:

$$u^*(t,x,\lambda) = \underset{u}{argmin} \left(L(x,u) + \lambda^T f(x,u) \right) = \underset{u}{argmin} H(x,u,\lambda) \tag{B.4}$$

Where the function H is called the Hamiltonian.

Then the question of how to obtain λ naturally arises. By differentiation of the HJB equation (B.2) and the boundary condition (B.3) with respect to x, we obtain:

$$-\dot{\lambda}^T = \frac{\partial}{\partial x}\left(H(x(t), u^*(t,x,\lambda), \lambda(t))\right)$$
$$\lambda(T)^T = \frac{\partial E}{\partial x}\left(x(T)\right)$$

By putting it all together, the initial optimization problem has been transformed to a *Boundary Value Problem* (i.e. differential equations with a set of boundary conditions that have to be satisfied):

$x(0)$	$=$	x_0		(initial value)
$\dot{x}(t)$	$=$	$f(x(t), u^*(t))$	$t \in [0,T]$	(ODE model)
$-\dot{\lambda}(t)$	$=$	$\frac{\partial}{\partial x}\left(H(x(t), u^*(t,x,\lambda), \lambda(t))\right)^T$	$t \in [0,T]$	(adjoint equations)
$u^*(t)$	$=$	$\underset{u}{argmin} H(x(t), u, \lambda(t))$	$t \in [0,T]$	(minimum principle)
$\lambda(T)$	$=$	$\frac{\partial E}{\partial x}\left(x(T)\right)^T$		(adjoint final value)

In order to obtain an explicit expression of the optimal control inputs (minimum principle) we have to minimize the Hamiltonian as B.4 suggests.A simple approach would be to set the derivative of the Hamiltonian with respect to u equal to zero and set additionally the condition that $\frac{\partial^2 H}{\partial u^2}$ is positive semidefinite. Without going into additional details, this problem can be solved using gradient, shooting or collocation methods. A flavor of shooting and collocation methods can be obtained in the next section. An overview of the different approaches for this can be found in [20] and a newer survey for applications of indirect methods for optimal control of rigid-link manipulators can be found in [21].
The punchline of the indirect methodology is 'first optimize, then discretize'. This leads to a relatively small boundary value problem that can sufficiently treat large scale systems. However, there are some weaknesses that make this approach less attractive, compared to direct methods. The necessary conditions apply only for *local* optimality. Additionally since an explicit expression of u^* is needed, singular arcs are difficult to treat. Another issue is that the ODE that we have to solve is strongly nonlinear and unstable which makes the solution difficult.

B.4 Direct Methods

The main idea of direct methods is to transform the initial problem into a *finite dimensional* non-linear programming(NLP), i.e. 'first discretize, then optimize'. The parametrization(discretization) can take place either in the control trajectory or in the state trajectory and this is the basic difference between the subgroups of direct methods that is going to be discussed in the following sub sections. A distinction that is made between the different methodologies is the notion of *sequential* or *simultaneous*. In sequential approaches the state trajectory $x(t)$ is regarded as an implicit function of the control inputs u(t) (or vice-versa) thus simulation and optimization proceed sequentially, one after the other. In contrast to this, simultaneous approaches keep a parametrization of the state trajectory as optimization variables within the NLP and add suitable equality constraints representing the ODE model. Thus, simulation and optimization proceed simultaneously and only at the solution of the NLP do the states actually represent a valid ODE solution corresponding to the control trajectory.

B.4.1 Single Shooting

In the single shooting approach we begin by discretizing the control inputs. A way to do this, is to choose grid points on the unit interval $0 = \tau_0 < \tau_1 \ldots < \tau_N = 1$ and then rescale

these grid points to the possible variable time horizon of the optimal control problem $[0, T]$, by defining

$$t_i = T\tau_i \ , \ i = 0, 1, \ldots, N$$

On this grid we then discretize the controls u(t), for example piecewise constant, $u(t) = q_i$ for $t \in [t_i, t_{i+1}]$ so that u(t) only depends on the finitely many control parameters $q = (q_0, q_1, \ldots, q_{N-1}, T)$ and can be denoted by $u(t, q)$. If the optimization problem has fixed horizon length T, the last component of q disappears as it is not an optimization variable then.
Now the states x(t) on $[0, T]$ can be regarded as dependent variables since their values can be obtained by using a numerical simulation routine to solve the initial value problem:

$$x(0) = x_0, \ \ \dot{x}(t) = f\left(x(t), u(t, q)\right) \ \ , \ \ t \in [0, T]$$

The selection of the ODE solver is crucial and the best choice depends on the model. The constraints values are also discretized using the same or finer grid. As a result, the following finite dimensional nonlinear programming problem is obtained:

$$J = \underset{q}{minimize} \quad \int_0^T L(x(t, q), u(t, q))dt + E(x(T, q))$$

Subject to:

$$h(x(t_i, q), u(t_i, q)) \geq 0, \quad i = 0, \ldots, N \quad \text{(discretized constraints)}$$

$$r(x(T)) = 0, \quad \text{(terminal constraints)}$$

This problem is solved by a finite dimensional optimization solver, e.g. Sequential Quadratic Programming(Section 2.6).
Notice that another starting point for direct single shooting could be to discretize the state trajectories (e.g. using B-Splines). Then by using a fixed control strategy (so that the control inputs become a function of the desired trajectory) the resulting optimization problem would be similar to the one we obtain by discretizing the control inputs. Therefore the methodology that was used in this project falls under this category.
The strong points of single shooting methods are:

- It can use fully adaptive, state-of-the-art ODE (for continuous time) or DAE (for discrete time) solvers
- It has only a few optimization degrees especially for higher order systems.
- Due its simplicity its very often used.

The weak points of single shooting methods are:

- The ODE solution $x(t, q)$ can depend 'very nonlinearly' on q. This reduces the performance of the optimization algorithm.
- Unstable systems are difficult to treat.

B.4.2 Collocation

We start by discretizing both the controls and the states on a *fine* grid. Typically the controls are chosen to be piecewise constant, with values q_i on each interval $[t_i, t_{i+1}]$ (similar to single shooting methods). The values of the states at the grid points will be denoted by $s_i = x(t_i)$. In collocation the ODE (infinite):

$$\dot{x}(t) - f\left(x(t), u(t)\right) = 0 \ \ , \ \ t \in [0, T]$$

is replaced by finitely many equality constraints

$$c_i(q_i, s_i, s_i', s_{i+1}) = 0 \ \ , \ \ i = 0, \ldots, N-1 \tag{B.5}$$

where the additional variables s_i' may represent the state trajectory on intermediate 'collocation points' within the interval $[t_i, t_{i+1}]$. By suitable choice of these points a high

approximation order can be achieved and typically they are chosen to be the zeros of orthogonal polynomials.[1] Assume for simplicity that no intermediate points s_i' are present. The additional equalities in B.5 are given by:

$$c_i(q_i, s_i, s_{i+1}) := \frac{s_{i+1} - s_i}{t_{i+1} - t_i} - f\left(\frac{s_i + s_{i+1}}{2}, q_i\right) \quad , \quad i = 0, \ldots, N-1$$

Then we also approximate the integrals on the collocation intervals, e.g. by

$$l_i(q_i, s_i, s_{i+1}) := L\left(\frac{s_i + s_{i+1}}{2}, q_i\right)(t_{i+1} - t_i) \approx \int_{t_i}^{t_{i+1}} L(x(t), u(t))dt \quad , \quad i = 0, \ldots, N-1$$

After the discretization we obtain the following problem:

$$\underset{s,q}{minimize} \quad \sum_{i=0}^{N-1} l_i(q_i, s_i, s_{i+1}) \; + \; E(s_N)$$

Subject to:

$s_0 - x_0 = 0,$		(fixed initial value)
$c_i(q_i, s_i, s_{i+1}) = 0,$	$i = 0, \ldots, N-1$	(discretized ODE model)
$h(s_i, q_i) \geq 0,$	$i = 0, \ldots, N$	(discretized constraints)
$r(s_N) = 0,$		(terminal constraints)

This is a *large scale* but *sparse* optimization problem.[2] It can be solved using SQP method for sparse problems or by an interior-point method.
The strong points of collocation are:

- A sparse NLP is obtained (faster to solve).
- Is shows fast local convergence
- It can treat unstable systems well.
- It can easily cope with state and terminal constraints.

The major weak point of collocation is that adaptive discretization error control needs to reproduce the grid and thus applications of collocation do not usually address the question of proper discretization error control.

B.4.3 Multiple Shooting

This method tries to combine the advantages of a simultaneous method, like collocation, with the major advantage of single shooting which is the ability to use adaptive, error controller ODE solvers. First, similarly to the previous direct methods, we discretize the control inputs and setting their value piecewise constant between the grid points:

$$u(t) = q_i \quad , \quad t \in [t_i, t_{i+1}]$$

The grid can be as large as in single shooting (and not 'fine' as in collocation). Then we solve the ODE on each interval $[t_i, t_{i+1}]$ independently, starting with an artificial initial value s_i:

$$\begin{aligned} \dot{x}_i(t) &= f(x_i(t), q_i) \quad , \quad t \in [t_i, t_{i+1}] \\ x_i(t_i) &= s_i \end{aligned}$$

[1] A sequence of polynomials is called orthogonal when any two different polynomials in the sequence are orthogonal to each other under *some* inner product. The zeros of orthogonal polynomials are simply the roots of the respective polynomials.

[2] An optimization problem is called sparse when the matrices that are produced in order to solve the problem contain a lot of zero entries. Sparse matrices lead to faster iterations for some optimization algorithms.

By numerical solution of these initial value problems we obtain trajectory pieces $x_i(t, s_i, q_i)$. Notice that the trajectory piece depends on the interval's initial values (s_i) and the control inputs (q_i). Alongside with the decoupled ODE solution we can also numerically compute the integrals:

$$l_i(s_i, q_i) := \int_{t_i}^{t_{i+1}} L(x_i(t_i, s_i, q_i), q_i)dt$$

In order to constrain the artificial degrees of freedom s_i to physically meaningful values we impose continuity conditions between the intervals:

$$s_{i+1} = x_i(t_i, s_i, q_i) \ \ \forall i$$

This way, we reach to a NLP formulation that is equivalent to the single shooting case but contains the extra variables s_i, and has a block sparse structure:

$$J = \underset{s,q}{minimize} \quad \sum_0^{N-1} l_i(s_i, q_i) + E(s_N)$$

Subject to:

$s_0 - x_0 = 0,$		(fixed initial value)
$s_{i+1} - x_i(t_{i+1}, s_i, q_i) = 0,$	$i = 0, \ldots, N-1$	(continuity)
$h(s_i, q_i) \geq 0,$	$i = 0, \ldots, N$	(discretized constraints)
$r(s_N) = 0,$		(terminal constraints)

The strong points of direct multiple shooting are:

- It can combine the adaptivity with fixed NLP dimensions, by using adaptive ODE/DAE solvers.
- Within each iteration the most costly part is often the ODE solution that is easy to parallelize.
- Compared to collocation the NLP is of smaller dimension (but less sparse).
- From a practical point of view it offers the advantage that the user doesn't have to decide about the grid size for the ODE discretization (due to the ability to use adaptive solvers) but only on the control grid.

The major weak point of multiple shooting is that due to the loss of sparsity and the cost of the ODE solver each iteration demands higher CPU time theoretically. However the usage of adaptive state-of-the-art ODE/DAE solvers can compensate for this higher computational cost by improving the quality of the optimization procedure for each iteration.

B.5 Non-Linear Model Predictive Control

The main idea in this approach is to postpone the design of the control policy until the 'last minute' and only find controls for the states that are actually visited. This is done by re-optimizing the movement trajectory and associated control sequence at each time step of the control loop, always starting at the current state estimate.
There is a significant difference between this approach compared to all the previous ones; there is not a single optimization problem anymore that we are solving but we have to sequentially re-solve it for every time step. So far we saw algorithms that tackle with the general optimization problem but none is tailored for so fast computations, so that we can re-run it in small fractions of time (e.g. milliseconds)! This leads us to the need of some 'approximately optimal control' strategies that in order to make this problem tractable are sacrificing precision of the system's dynamics and calculations related to the moving steps during optimization.
Although NLMPC has been successfully applied to other fields, where the dynamics are sufficiently slow and smooth (e.g. chemical process control), in robotics this approach

is still rarely used mainly because of the fast dynamics (this can make the predictions very poor). This is an active and promising field for the years to come. An approach that has already given some interesting results on this field ([23])is based on the *Iterative Linear Quadratic Gaussian trajectory optimizer (iLQG)*. The iLQG approach, introduced in [24], is meant to find 'locally optimal' feedback control laws by iteratively solving a local LQG problem. It should be noted that the iLQG approach can also be used for offline optimization (i.e. only run it once) as in [24], [25].

Bibliography

[1] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo: *Robotics: Modelling, Planning and Control* . Springer-Verlag London Limited, 2009.

[2] F. Lewis, D. Dawson, C. Abdallah: *Robotic Manipulator Control Theory and Practice*. 2^{nd} Edition, Marcel Dekker Inc, 2004.

[3] R. Murray, Z. Li, S. Shankar Sastry: *A Mathematical Introducion to Robotic Manipulation*. CRC Press, 1994.

[4] T. Kailath: *Linear Systems*. Prentice-Hall, 1980.

[5] P. Prenter: *Splines and Variational Methods*. Willey & Sons, 1989.

[6] P. E. Gill, W. Murray, M. H. Wright: *Practical Optimization*. Academic Press, 1981.

[7] R. Fletcher: *Practical Methods of Optimization*. Willey & Sons, 1987.

[8] R. Hoppe: *Optimization Theory Scribe notes*. `http://www.math.uh.edu/~rohop/fall_06/`, 2006.

[9] R. Johanson: *Quadratic Optimization of Motion Coordination and Control* . IEEE Transactions on Automatic Control Vol. 35, 1990.

[10] S. Thomson, R. Patel: *Formulation of Joint Trajectories for Industrial Robots Using B-Splines*. IEEE Transactions on Industrial Electronics. Vol. IE-34. NO. 2.,1987.

[11] F. Park, J. Bobrow, S. Ploen: *A Lie Group Formulation of Robot Dynamics*. International Journal of Robotics Research. Vol. 14 No. 6, pp. 609-618, 1995.

[12] B. Martin, J. Bobrow: *Minimum-Effort Motions for Open-Chain Manipulators with Task-Dependent End-Effector Constraints*. International Journal of Robotics Research. Vol. 18 No. 2, pp. 213-224, 1999.

[13] C. Wang, W. Timoszyk, J. Bobrow: *Payload Maximization for Open Chained Manipulators: Finding Weightlifting Motions for a Puma 762 Robot*. Portion of paper was presented at the 1999 IEEE ICRA.

[14] C. Wang: *Optimal Path Generation for Robots*. Ph.D. Thesis, University of California, Irvine, 2000.

[15] Y. Chen: *Solving Robot Trajectory Planning Problems with Uniform Cubic B-Splines*. Optimal Control Applications & Methods, Vol. 12, 247-262, 1991.

[16] M. Schlemer: *On-line Trajectory Optimization for Kinematically Redundant Robot-Manipulators and Avoidance of Moving Obstacles*. Proceedings of the 1996 IEEE, 1996.

[17] S. LaValle: *Planning Algorithms*. Cambridge University Press, 2006.

[18] A. Ata: *Optimal trajectory planning of manipulators: a review*. Journal of Engineering, Science and Technology Vol.2 No. 1 (p.32-54) , 2007.

[19] D. Bertsekas: *Dynamic Programming and Optimal Control*. Third Edition. Athena Scientific, 2005.

[20] O. Stryk, R. Bulirsch: *Direct and Indirect Methods for Trajectory Optimization*. Annals of Operations Research, 1992.

[21] R. Callies, P. Rentrop: *Optimal Control of Rigid-Link Manipulators by Indirect Methods*. GAMM-Mitteilungen Volume 31, Issue 1, pp. 27-58, 2008.

[22] M. Diehl, H. Bock, H. Diedam, P. Wieber: *Fast Direct Multiple Shooting Algorithms for Optimal Robot Control*. Fast Motions in Biomechanics and Robotics, 2005.

[23] Y. Tassa, T. Erez, E. Todorov: *Synthesis and Stabilization of Complex Behaviors through Online trajectory optimization*. Fast Motions in Biomechanics and Robotics, 2005.

[24] W. Li, E. Todorov: *Iterative linearization methods for approximately optimal control and estimation of non-linear stochastic system*.International Journal of Control 80(9):1439-1453, 2007.

[25] D. Braun, M. Howard, S. Vijayakumar: *Exploiting Variable Stiffness in Explosive Movement Tasks*. Robotics: Science and Systems, 2011.