KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Saving Energy Catching A Ball: An Application of Shrinking Horizon Optimal Control to A Robot Arm

Xiyu Fu

Thesis submitted for the degree of Master of Science in Mechanical Engineering

Thesis supervisors:

Prof. dr. ir. Jan Swevers
Prof. dr. ir. Goele Pipeleers

Assessors:

Dr. ir. Joris Gillis
Dr. ir Claus Claeys

Mentors:

Ir. Niels van Duijkeren
Ir. Armin Steinhauser

Academic year 2017 – 2018

Preface

I would like to thank everyone who helped me during the last year. Special thanks to Niels van Duijkeren who spent a lot of time saving me from theoretical and practical difficulties. I don't know where could I get without his help. Moreover, I would like to thank all my supervisors, assessors and mentors for their helpful advices. And I would like to thank all readers for reading this thesis. My sincere gratitude also goes to my families. I wouldn't be here without their support.

Xiyu Fu

Contents

Abstract

An energy optimal control scheme is designed for a ball catching robot. By using the decoupled approach, the OCP in time domain is reformulated into a convex problem in path parameter domain through two nonlinear transformations. This reformulated problem is used in a real-time optimization approach so that the unexpected changes in ball trajectory and robot states could be taken care of. To speed up the optimization, a coarse grid and a shrinking horizon are used in the solving process. A linear interpolation method is used to better approximate the nonlinear optimal control input based on system feedback over large time intervals. Two estimators are designed to find the corresponding path parameter from robot state feedback. Using a heuristic method, the catch point is determined. The path is chosen as a 4th order polynomial and its coefficients are used to minimize path changes caused by changing catch point. An attempt to find the optimal path is also introduced.

Simulation results showed the proposed scheme could track the path when the catch point, and hence the path, changes during operation. A comparison between time optimal control validates the effectiveness of the proposed scheme in saving energy. Different factors that influence the tracking error and feasibility are discussed. The initial value of system state plays an important role in both subjects. Because the constraints are only imposed on collocation points, a coarse grid might cause infeasible problem. And finally it is shown that an energy optimal trajectory is not necessarily the slowest trajectory.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

COP	Constrained Optimization Problem
DAE	Differential Algebraic Equation
DOF	Degree of Freedom
DH	Denavit-Hartenberg
EKF	Extended Kalman Filter
HRC	Receding Horizon Control
HJB	Hamilton-Jacobi-Bellman
MPC	Model Predictive Control
NLP	Nonlinear Programming
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
QP	Quadratic Programming
UKF	Unscented Kalman Filter

Symbols

a	Control input of the transformed problem
a_{opt}	Optimal value of a
b	System state of the transformed problem
b_{init}	Initial value of b
b_{opt}	Optimal b
c	Equivalent Coriolis matrix
E	Thermal Energy
g	Equivalent gravity matrix
m	Equivalent inertia matrix
q	Joint position
$\dot{q}$	Joint velocity
$\ddot{q}$	Joint acceleration
s	Path parameter
s_{now}	Current path parameter
T_{catch}	Catching time
u	Control input
x	System state
α	Slope of b
γ	Weighting factor
τ	Joint torque

Chapter 1

Introduction

1.1 Motivation and Goal

Robots are widely used in modern industry. They not only increase productivity greatly but also free human from tedious and tiring work. While industrial robots find their application in more and more fields, the energy consumption of industrial robots is taking a increasing part of total energy consumption in the industry. A survey in 2015 [3] shows that 8% of energy in production industry is eaten by robots. Actuators in robots are the major energy consumers, and hence the major source of energy loss. About 60% of energy loss are copper loss I^2Rt [4]. One the other hand, heat generated during robot operation, and thermal expansion caused by it, is also a threat to robot's accuracy. And thermal constraints limit the continuous operating capacity of robots. Therefore, minimizing thermal energy generation during robot operation is of vital importance in improving energy efficiency as well as robots performance.

There are various attempts to reducing thermal energy generation in robots such as usage of permanent magnetics, increasing wire cross section and improvement of power electronics. One interesting way is by introducing optimal motion planning. It requires no or few modifications in hardwares thus could be applied in a wide range of existing robots. Early works in this direction could be dated back as early as 1980s [5] [6]. And recent researches has implemented real-time optimization in robot systems thanks to the increasing computation power and novel schemes. For example, in [7], the authors developed an energy optimal point-to-point control system using MPC.

Besides point-to-point motion, there are other benchmark tasks for control techniques. One example is the ball catching task. It is a complex problem that involves interaction between robot perception, planning, control and mechanics. There are numerous robot systems designed for such a task [8] [9] [10] [11]. Developing an energy optimal control scheme for ball catching problem could be a good example in understanding energy optimal control.

The first goal of this project is to learn optimal control techniques and to use them to formulate the energy optimal control problem. Then a ball catching scheme is to be built around the energy optimal control problem. The next step is to validate the scheme in simulation and to discuss the results.

1.2 Outline

In the second chapter, the energy optimal control problem is formulated and different ingredients for describing and solving the ball catching problem is introduced. Chapter 3 gives an overview of a proposed scheme for ball catching task. Several supporting units in the scheme are also introduced in this chapter. In chapter 4, the method for solving the energy optimal control problem is described. Simulation results and discussions are given in chapter 5. And chapter 6 provides a conclusion of this thesis.

Chapter 2

Problem Statement

Catching a ball is a complex task so it needs to be divided into subtasks. These subtasks fall in three categories, namely perception, planning and control. Perception tasks include estimating the ball's trajectory and evaluating the robot's own states. Planning where to catch the ball is also a question to be answered. And how to control the robot moving from its initial position to the catch point energy optimally is the core problem to be solved in this project. These subtasks and ingredients for describing them are presented in this chapter.

In this chapter, different models used in this project , namely the dynamic model of a flying ball, the kinematic model of the robot IRB 120 and the dynamic model of the robot, are described. It is followed by a brief introduction to numerical optimal control. And then the energy optimal control problem in time domain is formulated. A simplified example of solving such a problem in time domain using multiple shooting method is given. Finally, the disadvantage of solving this problem in time domain is discussed.

2.1 Dynamic Model of the Flying Ball

A dynamic model of the flying ball is needed for ball position prediction and interception point determination. In early works about ball-catching, e.g. B.Hove and J.E.Slotine [8], a simple projectile motion model was usually used. The air drag was assumed negligible. To have a faster convergence and a more accurate estimation, air drag is also included in this thesis, which results in the following model[9]:

$$\dot{\vec{x}} = \vec{v} \tag{2.1}$$

$$\dot{\vec{v}} = \vec{g} - \alpha|\vec{v}|\vec{v} \tag{2.2}$$

where $\alpha = \frac{C_w A \rho}{2m}$, $C_w = 0.45$, A is the ball's characteristic area, ρ is the density of air and m is the mass of the ball. For a tennis, these values [9] are listed in Table 2.1

Figure 2.1 shows simulated trajectories with or without air drag for a typical throw with initial velocity $vel_{init} = [2,\ 0,\ 10]^T\ [m/s]$. The end position after 2 seconds are $[4,\ 0,\ 0.38]^T$

Table 2.1: Aerodynamic data of a tennis

$A\,[m^2]$	$\rho\,[kg/m^3]$	$m\,[kg]$	$\alpha\,[m]$
2×10^{-3}	1.293	0.05	1.14×10^{-2}

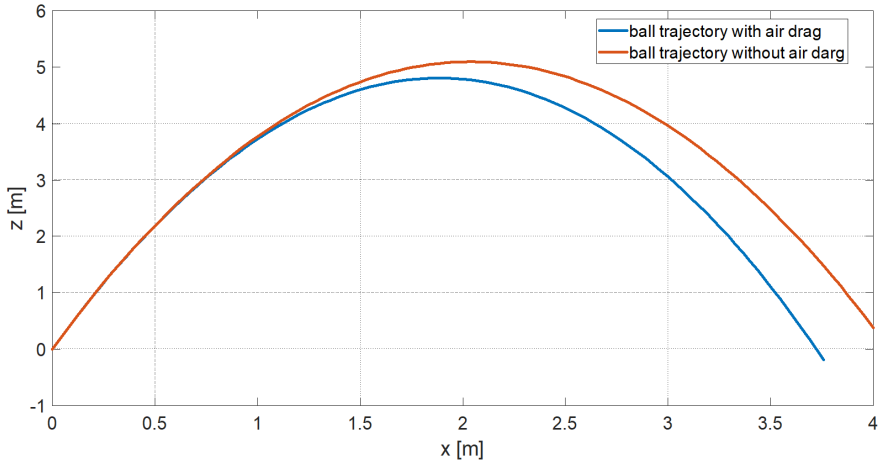


Figure 2.1: Simulated ball trajectories with or without air drag

for model without air drag and $[3.76,\ 0,\ -0.19]^T$ for model with air drag. The difference is more than 0.6 m which is a lot larger than the size of the ball or the catching glove. This results justifies the usage of a more complex model.

2.2 Kinematic Model of the Robot

The robot used in this project to catch the ball is the IRB 120 produced by ABB. IRB 120 is a compact 6-DOF robot arm. It has a nominal payload of 3 kg and a reach radius of 580 mm. More informations could be found on the website of ABB Robotics [2].

Forward Kinematics One compact way to describe the location and orientation of a rigid body with respect to a fixed coordinate frame is by using the special Euclidean group ($SE(3)$) [12]. To use this presentation, a coordinate frame is attached to the rigid body. And the transformation from the fixed frame to the attached frame (rigid body transformation) is described by a member in $SE(3)$ which could be written as a 4×4 matrix.

$$T=\left[\begin{array}{c|c}\boldsymbol{R} & \vec{p}\\ \hline \boldsymbol{0} & 1\end{array}\right]$$

The 3×3 matrix $\boldsymbol{R}$ indicates the rotation while the 3×1 vector $\vec{p}$ points from the fixed origin to the attached origin. For an open chain robot like the IRB 120, each of its link is attached to a coordinate frame and the ith frame is describe with respect to the $i-1th$ frame by a matrix T_i in $SE(3)$. To have the position and orientation of the end-effector

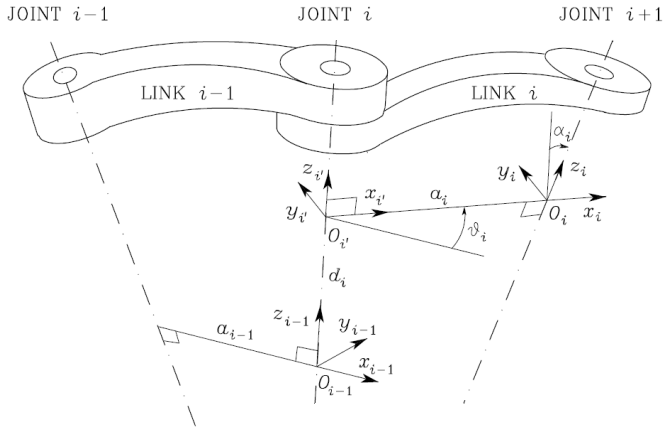


Figure 2.2: Denavit-Hartenberg convention [1]

with respect to the world frame, these matrices are multiplied one by one.

$$T_{ee}^{world} = \prod_{i=1}^{6} T_i^{i-1}(q_i) \tag{2.3}$$

There is a systematic way to determine frames attached to two consecutive links and the relative relationships between them, that is, the Denavit-Hartenberg convention. Figure 2.2 shows two consecutive links and their attached frames. The frame i could be determined by following the following procedure [1]:

- First find the axis of joint $i+1$ and choose it as z_i. Choose z_{i-1} in the same way.
- Draw the common normal of the above two axes. The intersection between z_i and the common normal is O_i, the origin of frame i. And O_i' is the intersection between z_{i-1} and the common normal.
- The x_i should be chosen along the common normal and its positive direction is pointing from O_i' to O_i.
- Choose y_i such that frame $O_i x_i y_i z_i$ is a right-hand frame. This completes the definition of the frame attached to link i.
- For frame 0, there is no z_0'. Therefore x_0 and O_0 are chosen arbitrarily.
- For the end-effector, there is no z_n since no other joint is connected to it. In this case, z_n is aligned with z_{n-1}.

By repeating the above procedure, all of the frames needed in the open chain are established. The next step is to describe the rigid transformation between these frames. There are four parameters (DH parameters) that are used to specify the transformation between frame $i-1$ and frame i [1]:

Table 2.2: DH parameters of IRB 120

Joint	$a\,[m]$	$d\,[m]$	$\alpha\,[rad]$	$\theta\,[rad]$ (offset)
1	0	0.29	$-\frac{\pi}{2}$	0
2	0.27	0	0	$-\frac{\pi}{2}$
3	0.07	0	$-\frac{\pi}{2}$	0
4	0	0.302	$\frac{\pi}{2}$	0
5	0	0	$-\frac{\pi}{2}$	0
6	0	0.21	0	π

- a_i, it is the length of $|O_i'O_i|$
- d_i, it is the distance between O_{i-1} and O_i'
- α_i, the angle by which rotating frame $i-1$ around x_i counter-clockwise will get parallel z_i and z_{i-1}.
- θ_i, the angle by which rotating frame $i-1$ around z_{i-1} will get parallel x_{i-1} and x_i.

All joints of IRB 120 are revolute joints and that makes θ_i the variable in DH parameters. By applying the DH convention to IRB 120, we could find its DH parameters as listed in Table 2.2 [13]. Building a kinematic model of a robot using DH parameters is a easy task in MATLAB with the help of the Robotics Toolbox designed by Peter Corke [14]. The result is shown in Figure 2.3.

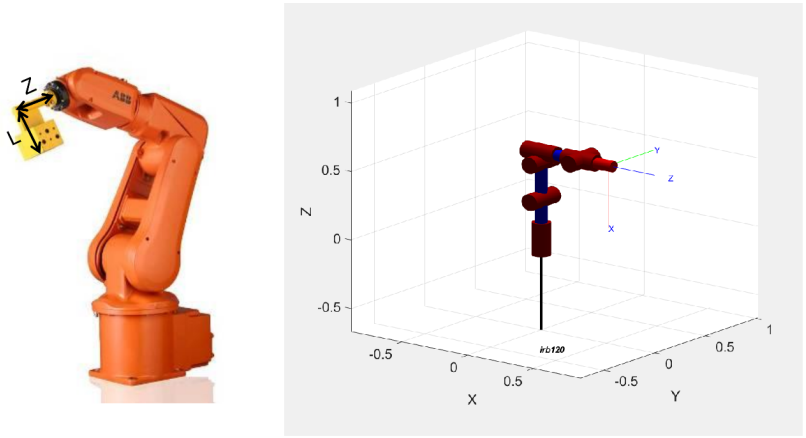


Figure 2.3: Left: IRB 120 [2] Right kinematic model of it

Inverse Kinematics The goal of inverse kinematics problem is to find the corresponding joint variables to a given end-effector position and orientation. Although there are some closed-form solutions for simple or special structures such as the three-link planar arm, the arm with spherical wrist and the anthropomorphic arm. There is no closed-form solution in general cases due to the highly nonlinear relationship between joint variables

and end-effector pose. It is necessary to resort to numerical method. Some well-known algorithms are the Jacobian (pseudo-)inverse, Jacobian transpose and their second order counterparts [1]. The Robotics Toolbox has already included several numerical algorithms for inverse kinematics. It is convenient to use them in MATLAB. In this project, the Jacobian (pseudo-)inverse algorithm is implemented in C++ for the IRB 120 model. This algorithm will be briefly introduced in the following paragraphs.

We first introduce the concept of *Analytical Jacobian*, $J_A(q)$. The position and orientation could be represented by a 6×1 vector $\vec{x}_e$. The upper three element is just the vector $\vec{p}$ in T_{ee}. The lower three element is the Euler angle between the end-effector's frame and fixed world frame. Let's denote the Euler angle by $\Phi_e = [\theta, \psi, \phi]^T$. It takes a few more calculation to get Φ_e from the rotation matrix R as shown in equation 2.4-2.6 [15].

$$\theta = -asin(R_{31}) \tag{2.4}$$

$$\psi = atan2(\frac{R_{32}}{cos\theta}, \frac{R_{33}}{cos\theta}) \tag{2.5}$$

$$\phi = atan2(\frac{R_{21}}{cos\theta}, \frac{R_{11}}{cos\theta}) \tag{2.6}$$

Taking the time derivative of $\vec{x}_e$, we will have:

$$\frac{d\vec{x}_e}{dt} = J_A \frac{d\vec{q}}{dt} \tag{2.7}$$

$$J_A = \frac{\partial \vec{x}_e}{\partial \vec{q}} \tag{2.8}$$

J_A is called the analytical Jacobian. It is a linear relationship between the joint velocity and the end-effector velocity. Usually J_A is a full rank matrix hence invertible. If it's not, the robot is either in singular position or in has redundancy. In either case, there are pseudo-inversion techniques to handle the problem. And we could solve the joint velocity by inverting J_A.

$$\frac{d\vec{q}}{dt} = J_A^{-1}(q) \frac{d\vec{x}_e}{dt} \tag{2.9}$$

One natural idea is to integrate equation 2.9 to solve the inverse kinematics problem. However, direct integration will introduce drift that is caused by numerical error. The longer the integration, the larger the error. This weakness could be overcame by resorting to the operational error:

$$\vec{e} = \vec{x}_d - \vec{x}_e \tag{2.10}$$

where $\vec{x}_d$ is the end-effector pose to be solved and $\vec{x}_e$ is the "current" pose calculated from forward kinematics. Taking the time derivative of equation 2.9, we get:

$$\dot{\vec{e}} = \dot{\vec{x}}_d - \dot{\vec{x}}_e \tag{2.11}$$

$$\dot{\vec{e}} = \dot{\vec{x}}_d - J_A(q)\dot{\vec{q}} \tag{2.12}$$

One could notice the similarity between these equations and equations of feedback control problems. From this similarity in mathematical form, the idea is design a feedback structure that make sure $\vec{x}_e$ converge to $\vec{x}_d$ asymptotically, just as what people do in feedback control.

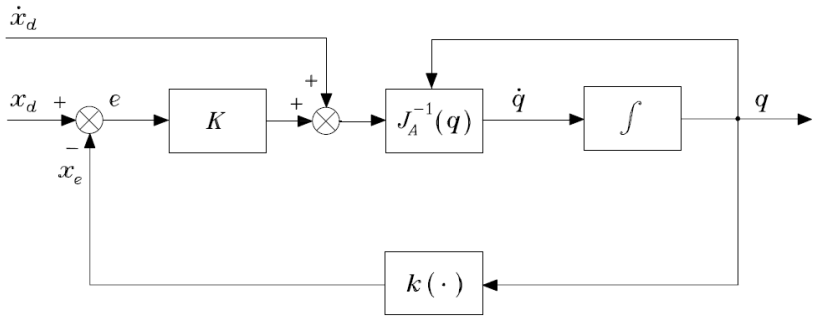


Figure 2.4: Block diagram of Jacobian inversion [1]

One simple choice is:

$$\dot{\vec{q}} = \boldsymbol{J}_A^{-1}(\boldsymbol{q})(\boldsymbol{q})(\dot{\vec{x}}_d + \boldsymbol{K}\vec{\boldsymbol{e}}) \tag{2.13}$$

which leads to the block diagram shown in figure 2.4. This scheme is called the Jacobian (pseudo-)inverse. K is a positive definite diagonal matrix. Note that K and $J_A^{-1}(q)$ are matrices hence they are linear operations. And the integration is also a linear operation. So the forward path is a linear system. In the feedback path, *k()* is the forward kinematics which is highly nonlinear. This scheme looks very similar to the proportional control scheme. The plant corresponds to the inversion of the analytical Jacobian and the observer or estimator is the forward kinematics. In the implementation of this project, $\dot{\vec{x}}_d$ is chosen as 0 and K is chosen to be a $\mathbf{6} \times \mathbf{6}$ identity matrix. Generally speaking, the larger the eigenvalues of K, the faster the convergence. But just like the case in feedback control, too large feedback gain results in unstable system. [1] gives an extensive introduction to inverse kinematics of robots including the above mentioned method.

2.3 Dynamic Model of the Robot

The goal of this project is to minimize the thermal energy generation during ball catching operations. Therefore, a dynamic model of the robot is necessary for motion planning and control. There are two commonly used method in dynamic modeling. The first one is based on the Lagrange's equations of the second kind. The second one uses the Newton-Euler formulation and the result is a recursive model. Although the second one is more efficient in terms of computation complexity, the first method is conceptually simple [1]. In the following sections, the Lagrange's equations will be used to model the dynamics of IRB 120 since it has only 6 DOF .

By using the Lagrange's equations, the dynamic model could be derived systematically. The first step is to choose a set of generalized coordinates and generalized forces. In robotics, they are usually chosen as the joint variables and joint torques (and reaction torque transmitted from the end-effector) respectively. Next thing to handle is to write down the kinematic energy and potential energy in terms of the generalized coordinates. The last step is filling in all these stuffs in the Lagrange's equations of the second kind.

This procedure is tedious and error-prone if it's performed by hand. Luckily there are tools to do it automatically, e.g., SymPyBotics [16]. The resulting model has the form:

$$\boldsymbol{\tau} - \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{h_e} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F_v}\dot{\boldsymbol{q}} + \boldsymbol{F_s}\boldsymbol{sgn}(\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) \tag{2.14}$$

$\boldsymbol{\tau}$ is the joint torque applied by the motor (through gear transmission). $\boldsymbol{h_e}$ is the force applied to the end-effector by the environment. It is neglected because there shouldn't be any contact with environment before the robot catches the ball. $\boldsymbol{M}(\boldsymbol{q})$ is the inertia matrix which is symmetric and positive definite. $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ represents the centrifugal and Coriolis effect. One notable property is that $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ is quadratic in $\dot{\boldsymbol{q}}$. We will utilize this property later. $\boldsymbol{F_v}\dot{\boldsymbol{q}}$ is the viscous friction and $\boldsymbol{F_s}\boldsymbol{sgn}(\dot{\boldsymbol{q}})$ is the Coulomb friction. $\boldsymbol{g}(\boldsymbol{q})$ refers to the gravity.

In order to identify the dynamic parameters of the robot (e.g., mass distribution, geometric parameters, etc), it's useful to find a linear relationship between these parameters and the torque applied by the motors. After reformulation (see section 7.2 in [1]), equation 4.30 could be written in the form:

$$\boldsymbol{\tau} = \boldsymbol{Y}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\boldsymbol{\pi} \tag{2.15}$$

$\boldsymbol{Y}$ is a function of joint position, joint velocity and joint acceleration. It is usually called the *regressor*. $\boldsymbol{\pi}$ is the set of dynamic parameters to be identified. For every joint, there are 13 parameters (including friction terms). These parameters are sometimes called the barycentric parameter. The linear relationship between barycentric parameters and joint torque, however, doesn't imply that this set of parameters is minimal. In other words, there are some parameters are redundant and might cause overfitting in identification. They could be further reduced into the so called base parameters [17]. There are 52 base parameters of IRB 120. They are identified by experiments.

The last part in the dynamic model section is to convert equation 2.15 into the form of 4.30. The friction terms are easy to handle because they could be found directly in the base parameters. Taking a look at equation 4.30, we could find that $\boldsymbol{M}(\boldsymbol{q})$ is proportional to $\ddot{\boldsymbol{q}}$. Therefore we have:

$$\boldsymbol{M}(\boldsymbol{q}) = \frac{\partial \boldsymbol{Y}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\boldsymbol{\pi}}{\partial \ddot{\boldsymbol{q}}} \tag{2.16}$$

Recall that $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ is quadratic in $\dot{\boldsymbol{q}}$, $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ could be found by:

$$\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \frac{1}{2}\frac{\partial}{\partial \dot{\boldsymbol{q}}}(\boldsymbol{Y}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\boldsymbol{\pi} - \boldsymbol{F_v}\dot{\boldsymbol{q}} - \boldsymbol{F_s}\boldsymbol{sgn}(\dot{\boldsymbol{q}})) \tag{2.17}$$

And by subtracting all other terms in equation 4.30, $\boldsymbol{g}(\boldsymbol{q})$ could be determined.

$$\boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{Y}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}) - \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} - \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \boldsymbol{F_v}\dot{\boldsymbol{q}} - \boldsymbol{F_s}\boldsymbol{sgn}(\dot{\boldsymbol{q}}) \tag{2.18}$$

These operations could be done automatically by CasADi [18] whose core function is automatic differentiation

2.4 Short Introduction to Optimal Control

The aim of optimal control is to find a control law (or a control policy, or a sequence of control history, etc) for a system such that a certain optimality criterion is achieved. Mathematically, it's to solve the following problem [19]:

$$\min_{u,x} \int_0^T L(x(t),u(t))dt + E(x(T),u(T)) \tag{2.19}$$

$$s.t. \quad x(0) = x_0 \tag{2.20}$$

$$\dot{x(t)} = f(x(t),u(t)) \tag{2.21}$$

$$h(x(t),u(t)) \geq 0 \tag{2.22}$$

$$r(x(T)) \geq 0 \tag{2.23}$$

$$t \in [0,T] \tag{2.24}$$

There are three families of methods to solve problem 2.19, namely the dynamic programming, the indirect method and the direct method.

Dynamic Programming This name refers to algorithms, not necessarily in optimization, that break down the original problem into easier sub-problems recursively and solve them. It was described by Richard Bellman as the 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." [20]

Basically it means that any sub-trajectory of an optimal trajectory is also optimal. If a value function (sometimes also called the cost-to-go) is defined as:

$$J(x(t_i),u(t_i),t_i) = \int_{t_i}^T L(x(t),u(t))dt + E(x(T),u(T)) \tag{2.25}$$

then the optimal solution should be the same as the solution to problem 2.19 in the time interval $t \in [t_i, T]$. In discrete time, this principle leads to the Bellman's equation:

$$J'(x_k,u_k,t_k) = \min_{x,u} L(x_k,u_k) + J'(f(x_k,u_k),u'_{k+1},t_{k+1}) \tag{2.26}$$

J' means optimal value function. This is a bottom-up approach. The first step is to solve for $t_N = T$. Then use the result to solve for t_{N-1}, then t_{N-2}, ... until t_0. In continuous time, Bellman's principle of optimality leads to the Hamilton-Jacobi-Bellman (HJB) Equation:

$$-\frac{\partial J'(x,u,t)}{\partial t} = \min_{u,x} L(x,u) + \frac{\partial J'(x,u,t)}{\partial x} f(x,u) \tag{2.27}$$

The disadvantage of dynamic programming is that high dimension partial differential equations are hard to solve. It's only used in some small scale systems.

Indirect Method In the indirect method, the optimal condition is derived from the Pontryagin's Minimum Principle:

$$u'(x,t,\lambda) = arg\min_{u} H(x,u,\lambda) \tag{2.28}$$

$$H(x,u,\lambda) = L(x,u) + \lambda^T f(x,u) \tag{2.29}$$

From HJB equation, the adjoint variable $\lambda(t)$ is the solution of the partial differential equation:

$$-\dot{\lambda(t)} = \frac{\partial H(x,u',\lambda)}{\partial x} \tag{2.30}$$

Together with the initial value and terminal value, we get a boundary value problem that could be solved numerically. However, this differential equation is highly nonlinear and unstable thus hard to solve.

Direct Method As described above, in the indirect method, the optimal condition of problem 2.19 is first derived. Then the boundary value problem is solved by numerical method. It's "first optimize then discretize". The direct method, on the contrary, is "first discretize then optimize". Continuous fucntion $x(t)$ and $u(t)$ are discretized in a time grid $t = [t_1, t_2, ..., t_{N+1}]$. The continuous cost function $J(x,u)$ which is a functional of $x(t)$ and $u(t)$ becomes a scalar valued function of $x = [x_1, x_2, ..., x_{N+1}]$ and $u = [u_1, u_2, ..., u_N]$. And the optimal control problem 2.19 is converted into a nonlinear programming problem (NLP). This is the most popular class of methods. Commonly used direct methods are the single shooting method, the multiple shooting method and the collocation method. In *the single shooting method*, the system is simulated from initial states using initial guess of control inputs u. This is done by an integrator, e.g., 4th order Runge-Kutta, that integrate the system dynamics $f(x,u)$. Then the decision variable u is optimized. For optimizing NLP, commonly used methods are active set method, interior point method, alternating direction method of multipliers, etc. In *the multiple shooting method*, the decision variable of the NLP is x and u. For each x_i and u_i where $i = 1, 2, ..., N$, the integrator will integrate $f(x,u)$ over the time interval $[t_i, t_{i+1}]$. The continuity of state trajectory is guaranteed by additional constraints $x_i = x_{i+1}$. The decision variables are then optimized.

The third commonly used direct method is called collocation method. It converts differential equations of system dynamics into algebraic equations on grid points. For example, the ordinary differential equation(ODE):

$$\dot{x} = f(x,u)$$

could be converted into:

$$\frac{x_{i+1} - x_i}{\Delta t} = f(x_{i+\frac{1}{2}}, u_i)$$

Depending how the ODE is converted, collocation method could be further divided into B-spline method, pseudo-spectral methods, etc. The collocation method is used in this project.

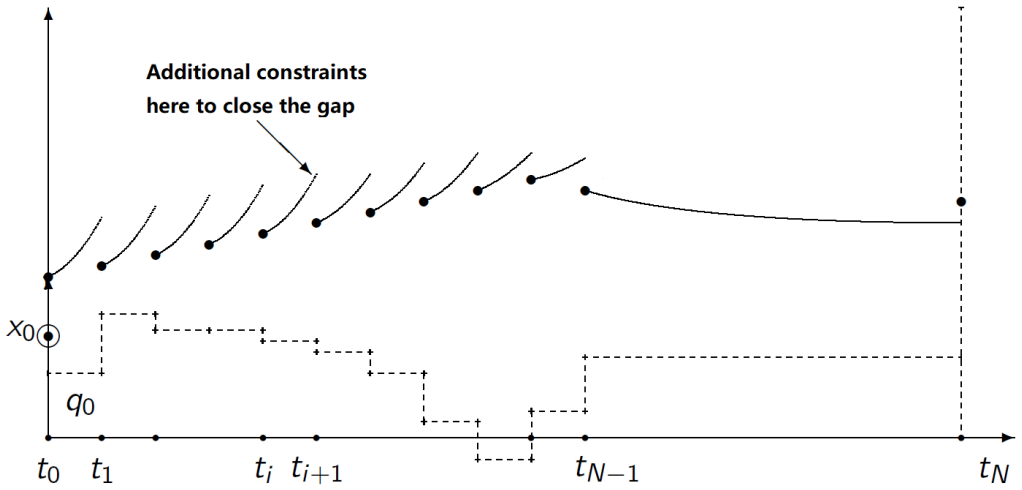


Figure 2.5: Multiple shooting method. Modified from lecture slides of M. Diehl

2.5 Formulation of the Energy Optimal Control Problem

As discussed in the introduction of this thesis, minimizing heat generation is very important in increasing robot accuracy, preventing unexpected failure and enlarging continuous operating limitation. To catch the ball in a thermal energy optimal way, the cost function of this optimal problem is chosen as:

$$\int_0^T \sum_{i=1}^{6} \frac{\tau_i^2(t)}{\tau_{max,i}^2} dt \tag{2.31}$$

The bridge that connects joint torque to thermal energy is the armature current:

$$\tau = K_\phi I \tag{2.32}$$

$$Q = RI^2 \tag{2.33}$$

Therefore, the square of joint torque is proportional to the thermal power of the motor. The states of the system are chosen as the positions and velocities of joints. $x = [q, \dot{q}]$. The constraints are:

System dynamics The system is model by the method mentioned in section 2.3.

$$\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_v\dot{q} + F_s sgn(\dot{q}) + g(q)$$

Time constraint The robot has to reach the interception point before the ball does.

$$0 \leq T \leq T_{catch}$$

Torque constraints The required torque can't be greater than the torque limits.

$$\tau_{min,i} \leq \tau_i \leq \tau_{max,i} \quad i = 1,2,...,6$$

Path constraints The robot follows a predefined geometric path $\boldsymbol{l(t)}$.

$$\boldsymbol{q_i = l(t_i)}$$

Boundary conditions The initial states and terminal states are fixed.

$$\boldsymbol{x(0) = x_{init}}$$
$$\boldsymbol{x(T) = x_{end}}$$

Combining the cost function with all of constraints above gives the optimal control problem to be solved:

$$\min_{x,T} \int_0^T \sum_{i=1}^{6} \frac{\tau_i^2(t)}{\tau_{max,i}^2} dt \tag{2.34}$$

$$s.t. \ \tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_v\dot{q} + F_s sgn(\dot{q}) + g(q) \tag{2.35}$$

$$0 \le T \le T_{catch} \tag{2.36}$$

$$\tau_{min,i} \le \tau_i \le \tau_{max,i} \quad i = 1,2,...,6 \tag{2.37}$$

$$q_i = l(t_i) \tag{2.38}$$

$$x(0) = x_{init} \tag{2.39}$$

$$x(T) = x_{end} \tag{2.40}$$

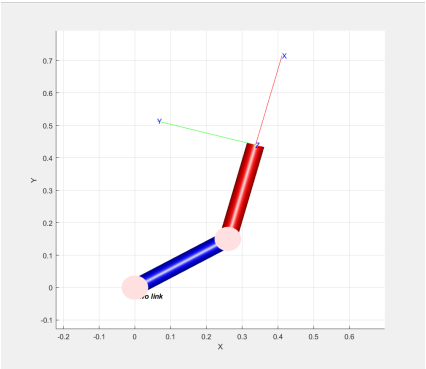
2.6 Solution to A Simplified Model in Time Domain

The first attempt is to solve this problem in time domain directly. Here, a simplified 2 DOF robot model is used. It has two links whose lengths are both 0.3 m. Their masses are 1.5 kg and 0.5 kg respectively. The centers of mass are assumed to be located in the middle of each link. Maximum joint torque is 10 Nm. The robot is asked to catch the ball along a parabolic path in workspace before t = 1.5 s. Figure 2.6a and 2.6b show the robot and the path in workspace.

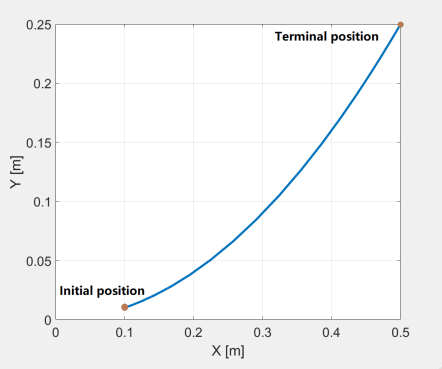
Table 2.3: DH parameters of a 2 DOF robot

Joint i	a_i	d_i	α_i	θ_i (offset)
1	0.3	0	0	0
2	0.3	0	0	0

The first step is to build the kinematic model of the robot. It's DH parameters are listed in Table 2.3. Next is to find the dynamic model. We don't need to identify any parameters in this example so the equation is directly written into the form of equation 4.30. It is



(a) Model of the 2 DOF robot



(b) Geometric path in workspace

assumed the friction is negligible.

$$\boldsymbol{\tau} = \begin{bmatrix} \mathbf{4.1575 + 0.09cos(q_2)} & \mathbf{1.0225 + 0.045cos(q_2)} \\ \mathbf{1.0225 + 0.045cos(q_2)} & \mathbf{1.0225} \end{bmatrix} \ddot{\boldsymbol{q}} + \begin{bmatrix} \mathbf{0} & \mathbf{-0.045\dot{q}_2 + 2\dot{q}_1 sin(q_2)} \\ \mathbf{0.045\dot{q}_1 sin(q_2)} & \mathbf{0} \end{bmatrix} \dot{\boldsymbol{q}} + \begin{bmatrix} \mathbf{5.886cos(q_1) + 1.4715cos(q_1 + q_2)} \\ \mathbf{1.4715cos(q_1 + q_2)} \end{bmatrix} \tag{2.41}$$

The method for solving the optimal control problem is chosen as the direct multiple shooting method. This problem is formulated in CasADi. Number of gird points $\boldsymbol{N}$ is set to 20. The solver for the NLP is the IPOPT (Interior Point OPTimizer) [21]. The ODE is integrated by the 4th order Runge-Kutta method. The number of iterations used to solve this problem is 188. IPOPT used 0.462 s to solve the NLP. The program was run on a Intel Core i7-6700HQ processor. The optimal trajectory and the optimal control history is shown in Figure 2.7

2.7 Conclusion

The optimal control problem to be solved is formulated in this chapter using models described above. A simplified version of this problem is solved in time domain by the multiple shooting method. Although the problem is solved successfully, it takes 188 iterations for the decision variables to converge. If the number of DOF of the robot increased to 6 like the IRB 120, computation cost for every iteration will be much higher. And the total time used to solve the problem might be longer than the catching time! On the other hand, the inertia matrix has to be inverted to get a first order ODE of states. In higher dimensions, this matrix will be much complex and it's not easy to find a analytical form of its inversion. More time is needed to invert $\boldsymbol{M(q)}$ numerically at each point. We need to find another method to solve the problem.

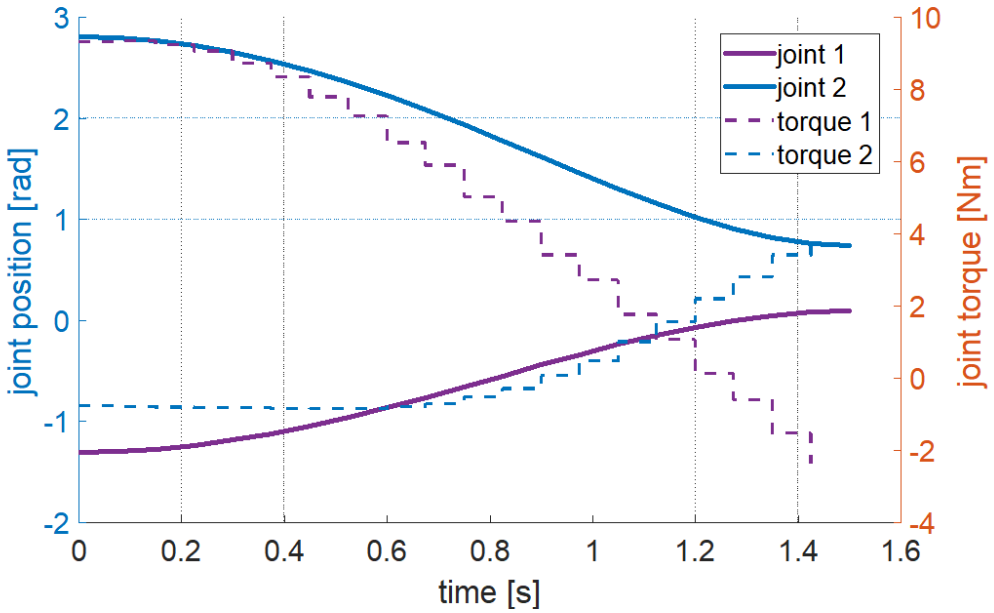


Figure 2.7: The optimal trajectory and control history

Besides solving the optimal problem, there are other problems. How to decide the interception point? How to change the geometric path if the estimation of ball trajectory is changed? In the next chapter, the structure of an algorithm to solve these problems will be introduced and details will be explained in chapter 4.

Chapter 3

Overview of the Scheme

In the previous chapter, the task catching a ball energy optimally is divided into subtasks. In this chapter, ideas of how to solve these subproblems are introduced. These ideas are combined together and a solution to the original problem is proposed. The flowchart of this solution is shown in Figure 3.1. This scheme contains one outer loop, which takes care of the ball estimation and planning iteratively, and one inner loop, that controls the robot's movement.

The first block is the ball trajectory estimation. It involves objects recognition, 3D extraction, parameter estimation and other subjects in computer vision. In a previous work [22], a vision system was developed for the same setup used in this project. That system used a monocular color camera and an EKF (Extended Kalman Filter) to estimate the ball's trajectory. Image processing is done with the help of OpenCV. The ball is recognized by its color (red) and its position in workspace is extracted based on passive monocular vision. However, the contrast of the image influences the accuracy a lot. An error of one pixel in ball recognition could result in centimeters deviation. Here, it is assumed a good estimator of the ball trajectory already exists and we could get the trajectory's parameters by making inquiries to it.

In the previous chapter, techniques for solving optimal control problems are introduced. The solution we get from these methods is a precomputed control trajectory. Applying this precomputed result to the robot is an open-loop control strategy. It is not a good idea since the Kalman filter estimates the ball continuously and updates the estimation with less and less uncertainty. Another disadvantage is caused by the so called model-plant-mismatch. Errors in the model will result in deviation from expectation. These are the motivations to solve the optimal control problem over and over again with the updated observation in real-time. Such an optimal feedback control via real-time optimization approach is known as Model Predictive Control (MPC) or Receding Horizon Control (RHC).

It is also mentioned before that solving the optimal control problem in time domain is computationally expensive when the number of DOF increases to 6. A decoupled approach with nonlinear transformation is used in this project. The resulting NLP is a

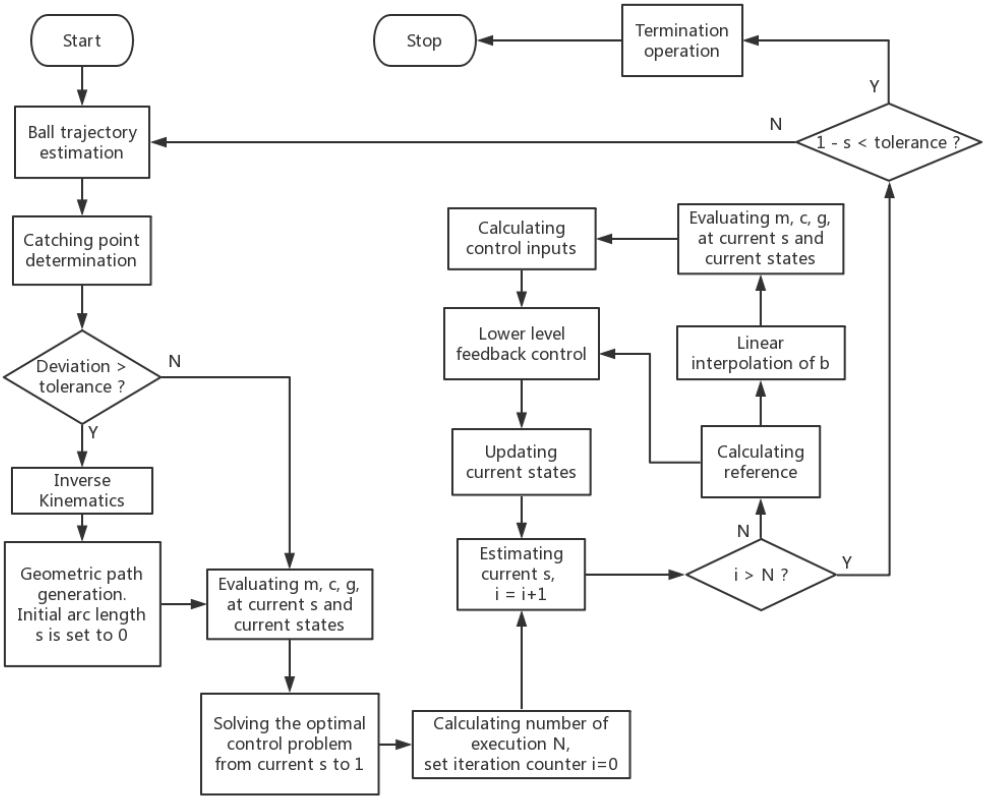


Figure 3.1: The flowchart of the scheme

convex problem. And the computation cost doesn't change as much as in time domain for a 6 DOF robot.

In this chapter, descriptions to blocks in the outer loop and the termination operation will be given. The blocks in the inner loop and the solution to the optimal control problem will be explained in next chapter.

3.1 Catch Point Determination

After knowing the ball's trajectory by consulting the estimator, a catch point should be determined. In most cases, the intersection between the trajectory of the ball and the workspace of the robot contains infinite points. Different methods have been used by researchers to determine at which point to catch the ball. In [10], a horizontal plane is placed at a given height and the catch point is the contact point between this plane and the ball's trajectory. In [23], it is the intersection with a vertical plane in front of the robot. The most complicated method is proposed in [11]. They have made the catch point's coordinate as a part of the decision variables and solved a very complex optimal

control problem. To get a smaller energy cost, the catch point should be close to the original position of the end-effector. And it shouldn't be too close to the robot itself to avoid awkward pose (that might be a singularity pose). In this project, a heuristic method proposed in [9] is used.

The first step is to find the point at which the ball flies into the workspace. Let's call it point $\boldsymbol{C}$. The ball's trajectory is simulated by an IDAS integrator [24] using parameters asked from the estimator. The outer surface of the IRB 120's workspace is a sphere so finding the point is simple. Then, the ball's trajectory is linearized at this point. The point on this line that is closest to the end effector could be found analytically by the following equations:

$$\boldsymbol{t} = \frac{\boldsymbol{x} - \boldsymbol{x_C}}{\boldsymbol{u}} = \frac{\boldsymbol{y} - \boldsymbol{y_C}}{\boldsymbol{v}} = \frac{\boldsymbol{z} - \boldsymbol{z_C}}{\boldsymbol{w}} \tag{3.1}$$

$$(\boldsymbol{x} - \boldsymbol{x_{ee}},\ \boldsymbol{y} - \boldsymbol{y_{ee}},\ \boldsymbol{z} - \boldsymbol{z_{ee}}) \cdot (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) = \boldsymbol{0} \tag{3.2}$$

$(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ is the velocity of the ball at point $\boldsymbol{C}$ (also the orientation of the tangent line). $(\boldsymbol{x_{ee}}, \boldsymbol{y_{ee}}, \boldsymbol{z_{ee}})$ is the original position of the end-effector. Let the point determined by equation 3.1 and 3.2 be point N. We could find the timestamp $\boldsymbol{t_N}$ of point N by following formula:

$$\boldsymbol{t_N} = \frac{\boldsymbol{u}(\boldsymbol{x_{ee}} - \boldsymbol{x_C}) + \boldsymbol{v}(\boldsymbol{y_{ee}} - \boldsymbol{y_C}) + \boldsymbol{w}(\boldsymbol{z_{ee}} - \boldsymbol{z_C})}{\boldsymbol{u^2} + \boldsymbol{v^2} + \boldsymbol{w^2}} \tag{3.3}$$

And the point in the original ball trajectory with the same timestamp $\boldsymbol{t_N}$ is chosen as the catch point $\boldsymbol{x_{end}}$. The procedure is illustrated in Figure 3.2

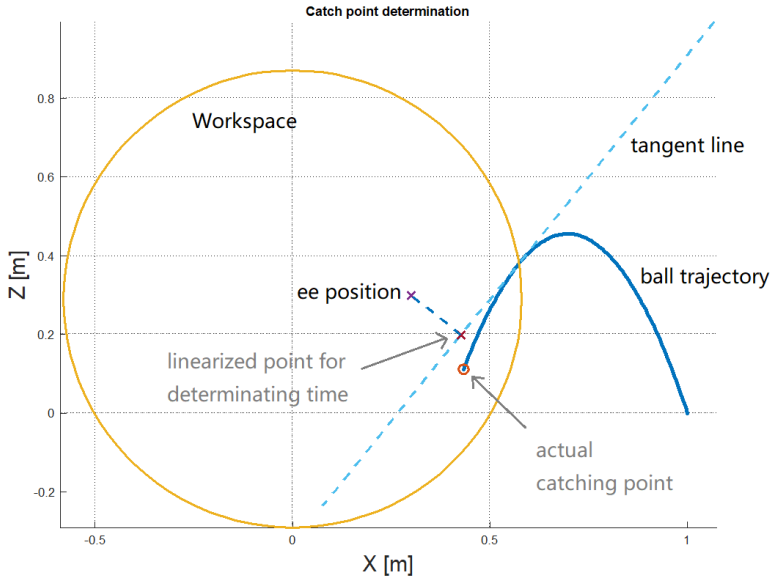


Figure 3.2: Determining the catch point

3.2 Geometric Path Generation

The decision block is used to check if there is a large deviation between the old catch point and the new catch point. It's possible that the catch point changes during looping due to the updated estimation. The Inverse Kinematics block has already been described in the previous chapter. Its output is the catch point in joint space $\boldsymbol{q_{end}}$. In the decoupled approach used in this project, a geometrical path $\boldsymbol{q(s)}$, that starts from current position of end-effector $\boldsymbol{q_{init}}$ to $\boldsymbol{q_{end}}$, is first planned. $\boldsymbol{s}$ is the arc length which ranges from 0 to 1. Then the mapping from time domain to arc length $\boldsymbol{s(t)}$ is established by solving the optimal control problem.

There are many algorithms that could find a collision-free path for the robot such as sample based algorithms like RRT (Rapidly-exploring Random Tree) [25], search based algorithm like A* and optimization based algorithm like CHOMP (Covariant Hamiltonian Optimization for Motion Planning) [26]. In this project, it is assumed no obstacles in the workspace (true in industrial environment, far from truth in daily life). Therefore obstacle avoidance is not a problem here. The geometric path is chosen to be a 4th order polynomial in joint space:

$$\boldsymbol{q(s) = a_0 s^4 + a_1 s^3 + a_2 s^2 + a_3 s + a_4} \tag{3.4}$$

$\boldsymbol{q(s)}$ subjects to the following constraints:

$$\boldsymbol{q(0) = q_{init}} \tag{3.5}$$

$$\boldsymbol{\frac{dq(0)}{ds} = 0} \tag{3.6}$$

$$\boldsymbol{q(1) = q_{end}} \tag{3.7}$$

$$\boldsymbol{\frac{dq(0)}{ds} = 0} \tag{3.8}$$

The reason for imposing 3.5 and 3.7 is obvious. Constraints 3.6 and 3.8 will be explained in chapter 4. Note that there are 5 coefficients in a 4th order polynomial so that there is still one degree of freedom. When the catch point changes (because the estimator gives new estimation), this extra DOF is used to generate a path that is closest to the old one by solving the following QP (Quadratic Programming Problem):

$$\boldsymbol{let \quad q_{new}(s) = c_0 s^4 + c_1 s^3 + c_2 s^2 + c_3 s + c_4} \tag{3.9}$$

$$\boldsymbol{\min_c \sum_{i=0}^{N} ||q_{new}(s_i) - q_{old}(s_i)||_2^2} \tag{3.10}$$

$$\boldsymbol{s.t. \quad q(0) = q_{previous}} \tag{3.11}$$

$$\boldsymbol{\frac{dq(0)}{ds} = \frac{dq_{previous}}{ds}} \tag{3.12}$$

$$\boldsymbol{q(1) = q_{end}} \tag{3.13}$$

$$\boldsymbol{\frac{dq(0)}{ds} = 0} \tag{3.14}$$

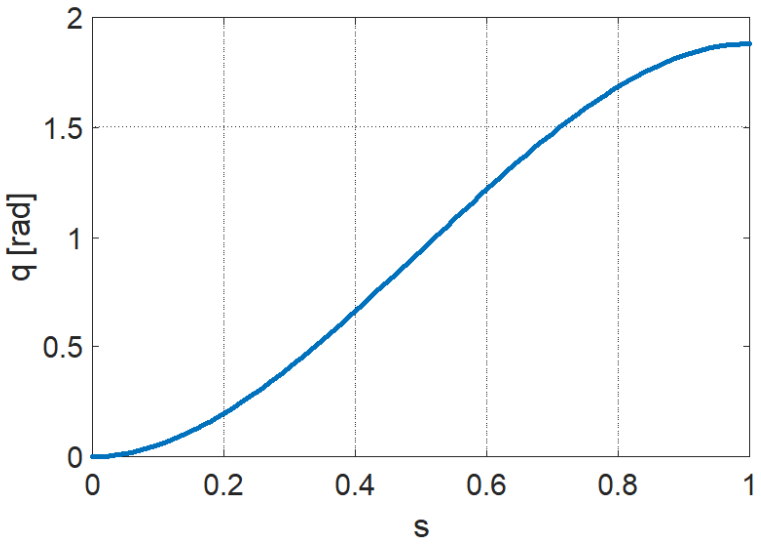


Figure 3.3: Example of $\boldsymbol{q(s)}$

One remark to constraint 3.12 and 3.14: This derivative could be easily written down analytically so we don't really have a differential constraint. Figure 3.3 shows an example of $\boldsymbol{q(s)}$. This topic is further discussed in the next chapter.

3.3 Terminal Operation

As mentioned in the title of this thesis, a shrinking horizon is used. A shrinking horizon means that the grid size of collocation method is becoming smaller and smaller as the robot moves to the catch point (see next chapter). When the robot is close to the catch point, the time interval of one execution might be smaller than the sample period of the robot ($\boldsymbol{4ms}$). It's not reasonable to keep working on real-time optimization since we can't apply the joint torque for, say, half a sample period. The terminal operation is used to control the robot moving to the catch point blindly using control history computed in the last iteration. Algorithm 1 describes how the terminal operation works.

The termination operation takes the current position, current velocity and optimal control history calculated as its inputs. $\boldsymbol{t_{sum}}$ accumulates $\boldsymbol{\Delta t(i)}$ until it is greater than sample period $\boldsymbol{T_s}$. Then an average of these torques are taken and set to the robot as control input. The reference velocity and reference position are calculated by linear interpolation. It is possible the last several control inputs is not applied because the summation of their time interval is still smaller than $\boldsymbol{T_s}$. So after the while loop, the reference velocity is set to 0, reference position is set to $\boldsymbol{q_{end}}$ and the reference torque is set to 0. This means that in the last sample period of the hold operation, the robot's torque control loop is disabled and it is control by PID control loop of joint position and velocity.

input: q_{end}, τ_{opt}, T_s, $q_{current}$, $\dot{q}_{current}$
set $t_{sum} = 0$, $\tau_{sum} = 0$, $n = 0$;
while $i < N+1$ **do**
  while $t_{sum} < T_s$ **do**
    $n++$;
    $i++$;
    $\tau_{sum} += \tau_{opt}(i)$;
    $t_{sum} += \Delta t(i)$;
  end
  $\tau_{avg} = \tau_{sum}/n$;
  $n = 0$;
  $\tau_{sum} = 0$;
  $t_{sum} = 0$;
  $q_{ref} = \frac{q_{end} - q_{current}}{N} i + q_{current}$;
  $\dot{q}_{ref} = \frac{-\dot{q}_{current}}{N} i + \dot{q}_{current}$;
  apply τ_{avg}, q_{ref}, $\dot{q}_{ref}$ to robot;
end
$\tau_{avg} = 0$, $q_{ref} = q_{end}$, $\dot{q}_{ref} = 0$;
apply τ_{avg}, q_{ref}, $\dot{q}_{ref}$ to robot;

Algorithm 1: Termination operation

This operation is suboptimal. But it only take a small part of the entire execution so the extra heat generation is also small. Figure 3.4 shows the last three time intervals and the corresponding average torque.

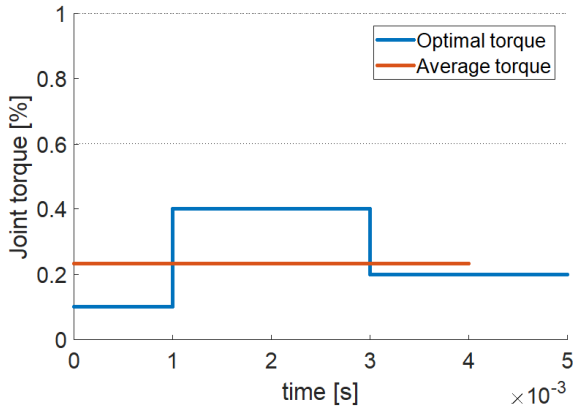


Figure 3.4: The last three time intervals

3.4 Conclusion

The overview of the scheme and some blocks have been described in this chapter. There is still one part about solving the optimal control problem that hasn't been discussed. It is done by two transformation, the first transforms the problem from time domain into path parameter domain and the second nonlinear transformation leads to a convex problem. The details of these two transformation and other unmentioned blocks will be described in the next chapter.

Chapter 4

Solving the Optimal Control Problem

It is a complex and difficult task to solve the optimal motion planning problem of a complex system like robots directly, i.e., trying to find an optimal path and an optimal trajectory at the same time [19] [27]. To get around this difficulty, a decoupled approach is proposed [28] [29]. In the decoupled approach, the problem is solved in two stages. First a geometric path is planned taking into account high level requirements like obstacle avoidance. Then a trajectory concerning system dynamics and actuators saturation is found.

The end-effector's motion along a geometric path could be described by two states, the path parameter s and its time derivative $\dot{s}$, instead of joint positions and velocities. Increasing DOF of the robot won't increase the computation cost. Therefore, transforming the problem from time domain into path parameter domain is preferred. Another advantage of solving the problem in path parameter domain is that we don't have to invert the inertia matrix of the robot. As shown in chapter 2, inverting $\boldsymbol{M(q)}$ is simple for a 2 DOF robot, but its complexity increases exponentially with DOF .

After transforming into path parameter domain, it is still not clear if the global optimal solution could be found. By taking a second nonlinear transformation as in [30], the problem could be reformulated into a convex problem so that any local optimizer is the global optimizer.

By solving the optimal control problem, a control history based on initial states is obtained. Applying this precomputed control history to the robot blindly (an open-loop control approach) is not a good idea. There are two reasons. 1) The ball's trajectory is estimated continuously so the catch point might change during the operation. And the later estimation is better than the former one (in terms of uncertainty) if a Bayesian filter like EKF or UKF is used. 2) The error in system model (model-plant-mismatch) can cause deviation from expected trajectory. To remedy this problem, the optimal control problem should be solved online. This is the idea of model predictive control (MPC).

However, for a system with high sample frequency ($T_s = 4ms$ in this project), solving the problem in time is not a easy task, especially for embedded controllers. To speed up the calculation, the number of collocation points N has to be limited. But small N gives raise to other problems. 1) Small N means large time intervals between collocation points, and optimal torques are only given at collocation points. The problem is that inside the time interval, optimal joint torque is nonlinear. For relatively long time intervals, the nonlinear torque shouldn't be approximated by just one value. 2) The second problem is the internal disadvantage of the collocation method. Constraints are only imposed on collocation points. To solve the first problem, a linear interpolation procedure is used to provide finer approximate control inputs based on online feedback. After solving the optimal control problem with a coarse grid, the optimal solution for the first time interval is taken and the linear interpolation computes close-loop control inputs for every sample period during the first time interval. After the first time interval is finished, the NLP is solved again. The second problem is discussed in chapter 5.

A shrinking horizon is also used in solving the problem. That is to use coarse grid at the beginning and a finer and finer grid during the operation. There are two reasons for doing so. First, the estimation of the ball's trajectory is improving with more and more data. So trying to have high accuracy by using fine grid during the entire operation is meaningless because the ball estimation is bad at the beginning. Second, even if the ball's trajectory is perfectly identified at first estimation, the robot don't have to move precisely along the path at the beginning. It only needs to move towards the catch point approximately. Precise movement is only required around the catch point.

In this chapter, the optimal control problem in time domain is transformed into the convex problem in path parameter domain. Then the linear predictor and the shrinking horizon are explained. And in the last part, two different path generators are described.

4.1 Transforming into Path Parameter Domain

The derivation in section 4.1-4.3 is based on [30]. As shown in chapter 2, the optimal control problem to be solved is:

$$\min_{x,T} \int_0^T \sum_{i=1}^{6} \frac{\tau_i^2(t)}{\tau_{max,i}^2} dt \tag{4.1}$$

$$s.t. \ \tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_v\dot{q} + F_s sgn(\dot{q}) + g(q) \tag{4.2}$$

$$0 \leq T \leq T_{catch} \tag{4.3}$$

$$\tau_{min,i} \leq \tau_i \leq \tau_{max,i} \quad i = 1,2,...,6 \tag{4.4}$$

$$q_i = l(t_i) \tag{4.5}$$

$$x(0) = x_{init} \tag{4.6}$$

$$x(T) = x_{end} \tag{4.7}$$

$\boldsymbol{x} = [\boldsymbol{q}\ \dot{\boldsymbol{q}}]^T$ is the robot states. Let $\boldsymbol{q}(\boldsymbol{s})$ be the path to be followed by the robot. $\boldsymbol{s}$ is the path parameter and it is normalized so that $\boldsymbol{s} \in [\boldsymbol{0}, \boldsymbol{1}]$. Using the chain rule, we have:

$$\dot{\boldsymbol{q}}(\boldsymbol{t}) = \frac{\boldsymbol{dq}}{\boldsymbol{ds}}\frac{\boldsymbol{ds}}{\boldsymbol{dt}} \tag{4.8}$$

$$\ddot{\boldsymbol{q}}(\boldsymbol{t}) = \frac{\boldsymbol{d^2q}}{\boldsymbol{ds^2}}(\frac{\boldsymbol{ds}}{\boldsymbol{dt}})^2 + \frac{\boldsymbol{dq}}{\boldsymbol{ds}}\frac{\boldsymbol{d^2s}}{\boldsymbol{dt^2}} \tag{4.9}$$

$$\boldsymbol{dt} = \frac{\boldsymbol{ds}}{\dot{\boldsymbol{s}}} \tag{4.10}$$

For simplicity, we denote $\frac{\boldsymbol{dq}}{\boldsymbol{ds}}$ as $\boldsymbol{q'(s)}$, $\frac{\boldsymbol{ds}}{\boldsymbol{dt}}$ as $\dot{\boldsymbol{s}}$, $\frac{\boldsymbol{d^2q}}{\boldsymbol{ds^2}}$ as $\boldsymbol{q''(s)}$ and $\frac{\boldsymbol{d^2s}}{\boldsymbol{dt^2}}$ as $\ddot{\boldsymbol{s}}$. The viscous friction $\boldsymbol{F_v\dot{q}}$ has to be neglected for the second transformation. Taking equation 4.8 and 4.9 into the problem and neglecting the viscous friction, we have:

$$\min_{\boldsymbol{\tau},\boldsymbol{s}} \int_0^1 \sum_{\boldsymbol{i}=1}^{6} \frac{\boldsymbol{\tau_i^2(s)}}{\boldsymbol{\tau_{max,i}^2}}\frac{\boldsymbol{1}}{\dot{\boldsymbol{s}}}\boldsymbol{ds} \tag{4.11}$$

$$\boldsymbol{s.t.}\ \boldsymbol{\tau} = \boldsymbol{m(s)}\ddot{\boldsymbol{s}} + \boldsymbol{c(s)}\dot{\boldsymbol{s}}^2 + \boldsymbol{g(s)} \tag{4.12}$$

$$\boldsymbol{0} \le \boldsymbol{T} \le \boldsymbol{T_{catch}} \tag{4.13}$$

$$\boldsymbol{\tau_{min,i}} \le \boldsymbol{\tau_i} \le \boldsymbol{\tau_{max,i}} \quad \boldsymbol{i = 1,2,...,6} \tag{4.14}$$

$$\boldsymbol{s(0) = 0} \tag{4.15}$$

$$\boldsymbol{s(T) = 1} \tag{4.16}$$

$$\dot{\boldsymbol{s}}\boldsymbol{(0)} = \dot{\boldsymbol{s}}_{\boldsymbol{0}} \tag{4.17}$$

$$\dot{\boldsymbol{s}}\boldsymbol{(t)} \ge \boldsymbol{0} \tag{4.18}$$

where

$$\boldsymbol{m(s)} = \boldsymbol{M(q(s))q'(s)} \tag{4.19}$$

$$\boldsymbol{c(s)} = \boldsymbol{M(q(s))q''(s)} + \boldsymbol{C(q(s),q'(s))q'(s)} \tag{4.20}$$

$$\boldsymbol{g(s)} = \boldsymbol{G(q(s))} + \boldsymbol{F_s sgn(q'(s))} \tag{4.21}$$

$$\boldsymbol{T} = \int_0^1 \frac{\boldsymbol{1}}{\dot{\boldsymbol{s}}}\boldsymbol{ds} \tag{4.22}$$

The last constraint $\dot{\boldsymbol{s}}\boldsymbol{(t)} \ge \boldsymbol{0}$ is imposed by the assumption that the robot won't go back along the path. This assumption is valid since going back and forth generates more thermal energy.

4.2 Reformulating to Convex Problem

By introducing the nonlinear transformation:

$$\dot{\boldsymbol{s}}^2 = \boldsymbol{b} \tag{4.23}$$

$$\ddot{\boldsymbol{s}} = \boldsymbol{a} \tag{4.24}$$

above problem could be reformulated as:

$$\min_{\tau,a,b} \int_0^1 \sum_{i=1}^{6} \frac{\tau_i^2(s)}{\tau_{max,i}^2} \frac{1}{\sqrt{b(s)}} ds \tag{4.25}$$

$$s.t. \ \tau = m(s)a + c(s)b + g(s) \tag{4.26}$$

$$0 \le T \le T_{catch} \tag{4.27}$$

$$\tau_{min,i} \le \tau_i \le \tau_{max,i} \quad i = 1,2,...,6 \tag{4.28}$$

$$b(0) = b_{init} \tag{4.29}$$

$$b'(s) = 2a'(s) \tag{4.30}$$

$$b(s) \ge 0 \tag{4.31}$$

Constraint 4.30 is obtained by combining the following two equations:

$$\dot{b} = b'(s)\dot{s} \tag{4.32}$$

$$\dot{b} = \frac{d(\dot{s}^2)}{dt} = 2\ddot{s}\dot{s} = 2a\dot{s} \tag{4.33}$$

This problem is convex because 1) the cost function is convex (integration preserves convexity) and 2) the constraints are linear.

4.3 Discretization by Collocation

If we regard s as a pseudo-time, then equation 4.30 becomes the system dynamics in common optimal control problems. The decision variable $a(s)$ is the s-domain counterpart of control input in time domain and $b(s)$ and $\tau(s)$ corresponds to system states. Problem 4.25-4.31 is solved by direct collocation method. Collocation points are equally spaced:

$$s_i = \frac{i}{N} \quad i = 0,1,2,...,N \tag{4.34}$$

$b(s)$ is evaluated at collocation points and the control input $a(s)$ is evaluated in the middle, i.e., $s_{i+\frac{1}{2}}$. The t-domain control input τ is also evaluated in the middle of collocation points. $\sqrt{b(s)}$ is approximated by:

$$\sqrt{b(s_{i+\frac{1}{2}})} = \frac{\sqrt{b_i} + \sqrt{b_{i+\frac{1}{2}}}}{2} \tag{4.35}$$

And the dynamic constraint 4.30 is approximated by simple finite difference method:

$$b_{i+1} - b_i = 2a_i \Delta s_i \tag{4.36}$$

After discretization, the NLP to be solved is:

$$\min_{\tau,a,b} \sum_{i=0}^{N-1} \sum_{j=1}^{6} \frac{\tau_j^2(s_{i+\frac{1}{2}})}{\tau_{j,max}^2} \frac{2}{\sqrt{b_{i+1}} + \sqrt{b_i}} \Delta s_i \tag{4.37}$$

$$s.t.\ \tau(s_{i+\frac{1}{2}}) = m((s_{i+\frac{1}{2}})a_i + c((s_{i+\frac{1}{2}}) \frac{b_i + b_{i+1}}{2} + g((s_{i+\frac{1}{2}}) \tag{4.38}$$

$$\tau_{min} \leq \tau \leq \tau_{max} \tag{4.39}$$

$$b_0 = b_{init} \tag{4.40}$$

$$b_{i+1} - b_i = 2a_i \Delta s \tag{4.41}$$

$$b(s) \geq 0 \tag{4.42}$$

$$0 \leq T = \sum_{i=0}^{N} \frac{2}{\sqrt{b_{i+1}} + \sqrt{b_i}} \Delta s_i \leq T_{catch} \tag{4.43}$$

Since $a(s)$ is regarded as the control input of this problem, it is assumed that a_i is constant within (s_i, s_{i+1}). As a consequence, from equation 4.30 we can find that $b(s)$ is linear within (s_i, s_{i+1}). $m((s_{i+\frac{1}{2}})$, $c((s_{i+\frac{1}{2}})$ and $g((s_{i+\frac{1}{2}})$ are parameters depending on the geometric path. In equation 4.38, joint torques τ is a multi-variable function of a and b, so τ_i is usually nonlinear on (s_i, s_{i+1}) as illustrated in Figure 4.1

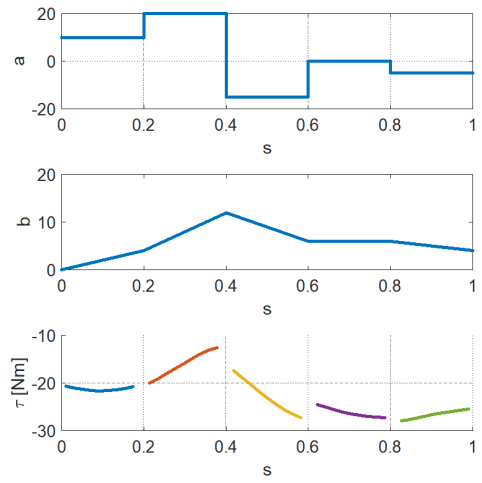


Figure 4.1: a is piecewise constant, b is piecewise linear and τ is piecewise nonlinear

4.4 Online Feedback and Linear Interpolation

As discussed above, changing catch point and model-plant-mismatch make the online feedback control necessary. The scheme is shown in Figure 4.2. As shown in the figure, reconfigurations of NLP are caused by two feedback paths. The first is the result of robot's movement and the second is the changing catch point.

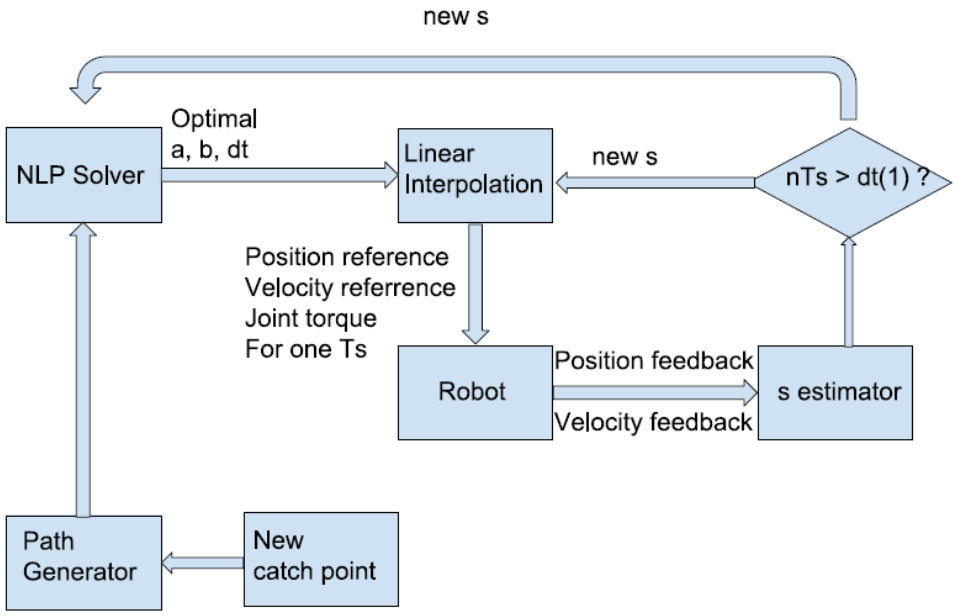


Figure 4.2: Online feedback scheme

Changes in catch point influence the optimal control by reconfiguring parameters in the NLP. Based on current catch point (hence current path) and current robot states, $\boldsymbol{m(s)}$, $\boldsymbol{c(s)}$ and $\boldsymbol{g(s)}$ in equations 4.19-4.21 are evaluated at collocation points. These data together with a new catch time $\boldsymbol{T_{catch}}$ are used to reconfigure the NLP . From the solution to NLP, optimal variables for the first time interval $\boldsymbol{\Delta t(1)}$ (corresponds to $[\boldsymbol{s_1}, \boldsymbol{s_2}]$ in s-domain), $\boldsymbol{a_{opt}(1)}$, $\boldsymbol{b_{opt}(1)}, \boldsymbol{b_{opt}(2)}$ are send to the linear interpolation block.

The linear interpolation block is used to provide optimal joint torques to the robot in every sample period. The gird size $\boldsymbol{\Delta s}$ is chosen to be large to speed up the calculation of NLP. And optimal torques are only given at midpoints of s intervals. As shown in Figure 4.1, optimal torques are nonlinear in one $\boldsymbol{s}$ interval (also nonlinear in one $\boldsymbol{t}$ interval). Therefore to move the robot more accurately, optimal joint torques are interpolated. As shown in Figure 4.1, $\boldsymbol{a(s)}$ is regarded as control input so it's constant during the first time interval. The first optimal solution of $\boldsymbol{a(s)}$, denoted as $\boldsymbol{a_{opt}(0)}$, is used in the interpolation. $\boldsymbol{b(s)}$ is linear due to constant $\boldsymbol{a(s)}$ and constraint 4.30. It's value for one specific sample period is calculated using current path parameter $\boldsymbol{s_{now}}$:

$$\boldsymbol{b_{now}} = \frac{\boldsymbol{b_{opt}(2)} - \boldsymbol{b_{opt}(1)}}{\boldsymbol{\Delta s(1)}}(\boldsymbol{s_{now}} - \boldsymbol{s_0}) + \boldsymbol{b_{opt}(0)} \tag{4.44}$$

The estimation of current path parameter $\boldsymbol{s_{now}}$ based on feedback data will be introduced later. Using the feedback data, we can get $\boldsymbol{q_{now}}$ and $\frac{\boldsymbol{dq}}{\boldsymbol{ds}}|_{\boldsymbol{now}}$. Note that $\boldsymbol{b}$ is defined as

$b = \dot{s}^2$, we have:

$$\frac{dq}{ds}|_{now} = \frac{\dot{q}}{\sqrt{b_{now}}} \tag{4.45}$$

With these data, $m(q_{now})$, $c(q_{now}, q'_{now})$ and $g(q_{now})$ could be computed using the robot's dynamic model and equation 4.19-4.21. The last step is to calculate control input, it's just a linear combination:

$$\tau = m(q_{now})a_{opt}(1) + c(q_{now}, q'_{now})b_{now} + g(q_{now}) \tag{4.46}$$

Another function of interpolation block is to calculate position and velocity reference for lower level controller. The integrated lower level controller of IRB 120 is shown in Figure 4.3. The question is how to find the expected position and velocity after current sample period without actually run a simulation. We know that ideally the robot follows the geometric path perfectly. So all we need to do is to find a relationship between sample period and increment in path parameter s. Start from the definition of b:

$$b = \dot{s}^2 = (\frac{ds}{dt})^2 \tag{4.47}$$

Discretize it at the midpoint between kth and $k+1$th collocation point:

$$\Delta t_k = \frac{2\Delta s_k}{\sqrt{b_{k+1}} + \sqrt{b_k}} \tag{4.48}$$

Utilize the piecewise linearity of b, replace b_{k+1} with $b_k + \alpha\Delta s_k$, $\alpha = \frac{b_{k+1}-b_k}{\Delta s_k}$ is a constant in this time interval. And let $\Delta t_k = T_s, b_k = b_{inter}$, the result is a equation of Δs:

$$T_s(\sqrt{b_{inter}} + \sqrt{\alpha\Delta s + b_{inter}}) = 2\Delta s \tag{4.49}$$

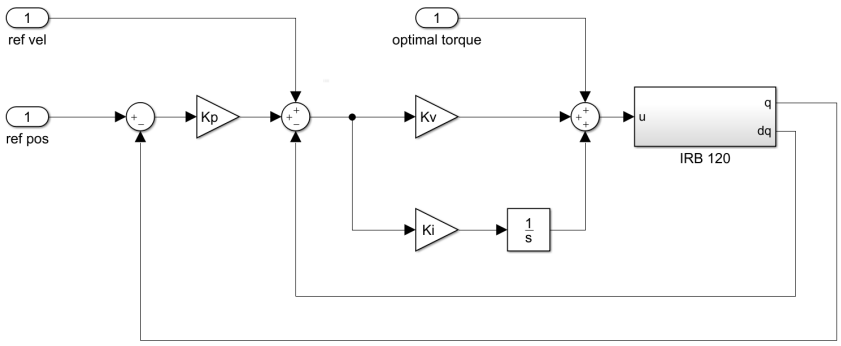


Figure 4.3: Lower level PID controller

Δs could be solved easily from this equation

$$\Delta s = T_s\sqrt{b_{inter}} + \frac{\alpha T_s^2}{4} \tag{4.50}$$

And the reference velocity and position is:

$$q_{ref} = q(s_{now} + \Delta s) \tag{4.51}$$
$$\dot{q}_{ref} = \dot{q}(s_{now} + \Delta s) \tag{4.52}$$

The scheme of linear interpolation block is summarized in algorithm 2

input: $a_{opt}(1),\ b_{opt}(1),\ b_{opt}(2),\ \Delta t(1),\ \Delta s(1),\ , s_0,\ T_s$
$n = \lfloor \frac{\Delta t(1)}{T_s} \rfloor$;
$i = 0$;
$\alpha = \frac{b_{opt}(2) - b_{opt}(1)}{\Delta s(1)}$;
while $i < n$ **do**
  read current states $q_{now}, \dot{q}_{now}$;
  estimate current path parameter s_{now};
  $b_{inter} = \alpha(s_{now} - s_0) + b_{opt}(1)$;
  $\frac{dq}{ds}|_{now} = \dot{q}_{now}/\sqrt{b_{inter}}$;
  evaluate $m(q_{now}), c(q_{now}, \frac{dq}{ds}|_{now}), g(q_{now})$;
  $\tau_{inter} = m\, a_{opt}(1) + c\, b_{inter} + g$;
  ;
  $\Delta s = T_s\sqrt{b_{inter}} + \frac{\alpha T_s^2}{4}$;
  $q_{ref} = q(s_{now} + \Delta s)$;
  $\dot{q}_{ref} = \dot{q}(s_{now} + \Delta s)$;
  ;
  send $\tau_{inter}, q_{ref}, \dot{q}_{ref}$ to robot
end

Algorithm 2: Linear Interpolation

4.5 Path Parameter Estimation

In section 4.4, the feedback of robot states is given in two ways. The first is by evaluating system dynamics $m(q),\ c(q, q'),\ g(q)$ at current state. The second is by estimating the corresponding path parameter s_{now}. A path estimator is used to find this parameter. Two path parameter estimators are introduced. The first one is used for continuous path and the second is for discrete path.

4.5.1 Estimating continuous path

This parameter is estimated by solving a nonlinear programming problem:

$$\min_{s} \quad ||q(s) - q_{now}||_2^2 + \gamma ||q'(s) - \frac{dq}{ds}|_{now}||_2^2 \tag{4.53}$$

$$s.t. \quad s_{old} - \delta s \leq s \leq s_{old} + \delta s \tag{4.54}$$

$$0 \leq s \leq 1 \tag{4.55}$$

We know that $q(s)$ is a 4th order polynomial and $q'(s)$ could be derived explicitly, although the cost function is nonlinear, it could be solved efficiently. The search of s is limited in the vicinity of previous path parameter s_{old}, the radius of this vicinity is δs. Here, it is set to $\frac{1}{N}$. Because $q(s)$ and $q'(s)$ has different unit, there is a weighting factor γ in the cost function. From experience, this weighing factor should be small, e.g. 10^{-6}. A good estimation of s_{now} is very important. Sometimes a bad estimation causes a significant lag behind the reference and eventually makes a feasible problem infeasible.

4.5.2 Estimating discrete path

In discrete situation, the problem becomes a little more difficult. During the operation, there is no guarantee that the robot's joint coordinates will locate at points in the trajectory. In fact, during the linear interpolation, the robot moves inside the gap in the discrete trajectory. And we can't just used the closest point in the trajectory. Even for a dense trajectory with total number of points $N_t = 999$, the robot's position of two consecutive sample periods might be close to the same point. And the estimator will believe that the robot didn't move at all.

A two step strategy is used in estimating discrete path. First step is to find the closet points in the trajectory. Let q_i and s_i denote the ith point in the trajectory and the corresponding path parameter:

$$\min_{s_i} \quad ||q_i - q_{now}||_2^2 \tag{4.56}$$

This search could be done in linear time $o(N_t)$. Let's call the results q_i* and s_i*. Next, linearize the trajectory in the vicinity of (s_i*, q_i*) to get a continuous straight line $q_l(s)$ and conduct search on it.

$$q_l(s) = \frac{q_{i+1} - qi - 1}{s_{i+1} - s_{i-1}}(s - s_i*) + q_i* \tag{4.57}$$

$$\min_{s} \quad ||q_l(s) - q_{now}||_2^2 \tag{4.58}$$

$$s.t. \quad s_{i-1} \leq s \leq s_{i+1} \tag{4.59}$$

$$1 \leq s \leq N_t \tag{4.60}$$

Let's call the result $s*$. To use $s*$ in the feedback, the trajectory is linear interpolated around s_i*:

$$q* = \frac{q_{i+1} - qi - 1}{si + 1 - si - 1}(s* - s_i*) + q_i* \tag{4.61}$$

The scheme is shown in Figure 4.4.

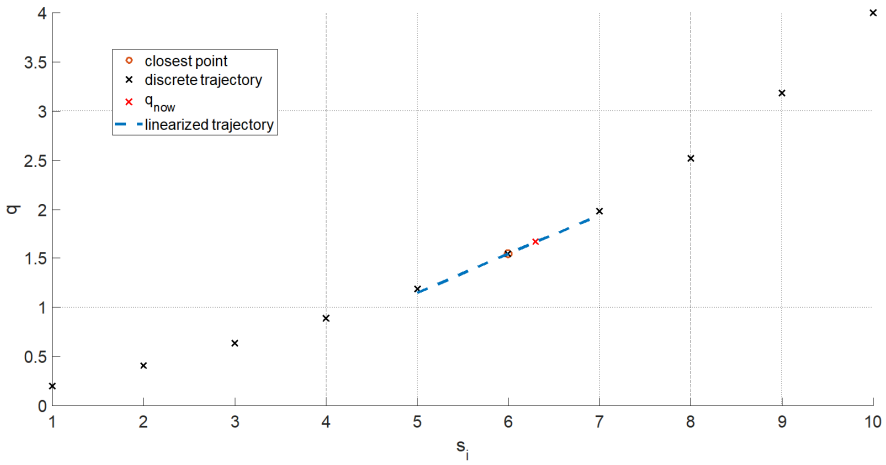


Figure 4.4: Estimating discrete path parameter

4.6 Shrinking Horizon

To catch a ball,the robot's movement doesn't need to be very accurate at the beginning. High precision is needed only when the robot is close to the catch point. So we could make the gird coarse at the beginning and let the linear interpolation do its job. And the number of collocation points could be small so that speed of solving the NLP is faster. While the robot is moving towards the catch point, the gird becomes finer and finer. How to get such a shrinking horizon? One idea is to reduce the number of collocation points $\boldsymbol{N}$. However, reducing $\boldsymbol{N}$ means reformulating the NLP in CasADi which takes a long time (reallocation of memory). An alternative method is based on the fact that the remaining path length is shorter and shorter. If the grid size is linked to the remaining path length, a shrinking horizon will be obtained with a fixed $\boldsymbol{N}$. This could be done by modifying the cost function:

$$\int_{s_{now}}^{1} \sum_{i=1}^{6} \frac{\tau_i^2(s)}{\tau_{max,i}^2} \frac{1}{\sqrt{b(s)}} ds \tag{4.62}$$

Instead of integrating from $\mathbf{0}$ to $\mathbf{1}$, the integration begins from the path parameter estimated from current states. And the grid size is:

$$\Delta s = \frac{1 - s_{now}}{N - 1} \tag{4.63}$$

As the robot approaching the catch point, s_{now} becomes closer and closer to 1. And the grid size is shrinking with a fixed $\boldsymbol{N}$ as shown in Figure 4.5

4.7 Path Generation

The optimal control problem discussed above is solved with an assumption that a geometric path has already be given. Now the question is how to find this path. First, we

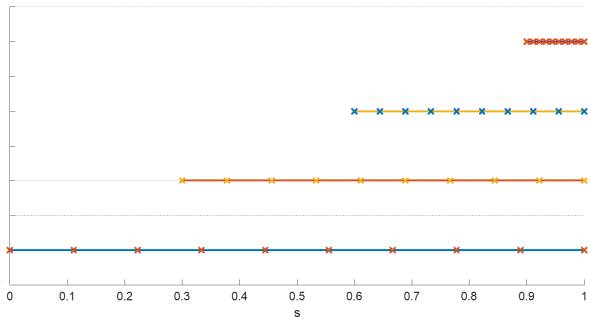


Figure 4.5: Shrinking horizon, N = 10

need to consider constraints imposed on this path. The most obvious constraint is the end position which has to be the catch point:

$$\boldsymbol{q}(\mathbf{1}) = \boldsymbol{q}_{\boldsymbol{catch}} \tag{4.64}$$

and the robot starts from it initial position:

$$\boldsymbol{q}(\mathbf{0}) = \boldsymbol{q}_{\boldsymbol{init}} \tag{4.65}$$

The robot should stop at the catch point, therefore we should put an additional velocity constraint in problem 4.25-4.31:

$$\frac{\boldsymbol{dq}}{\boldsymbol{dt}}|_{\boldsymbol{end}} = \mathbf{0} \tag{4.66}$$

But such a constraint can cause numerical difficulties in solving the NLP. If the $\boldsymbol{q}'(\boldsymbol{s})$ is not 0 at the end, $\boldsymbol{b}(\mathbf{1})$ has to be 0 to meet this constraint. And $\boldsymbol{b} = \mathbf{0}$ will cause a 'Invalid Number' error in CasADi. It is because the automatic differentiation decomposes a function into a chain of elementary operations and somewhere in this chain, $\boldsymbol{b}$ is the denominator. So a 0 leads to infinite. Therefore, to make the robot stop at the end, a constraint is imposed on the path:

$$\boldsymbol{q}'(\mathbf{1}) = \mathbf{0} \tag{4.67}$$

For the same reason, if the robot starts from stand-still, it's velocity needs to be 0. And the constraint to model the stand-still is:

$$\boldsymbol{q}'(\mathbf{0}) = \mathbf{0} \tag{4.68}$$

4.7.1 Continuous path

There are 4 constraints on the path. So if we wants to describe a path using functions with analytical expression, these function should have at least 4 parameters to be adjusted. And there is another requirement for the path. It can't be too long otherwise the problem becomes infeasible (see discussion in next chapter). One simple function that could meet all the requirement is a 3rd order polynomial function. It is determined by above constraints.

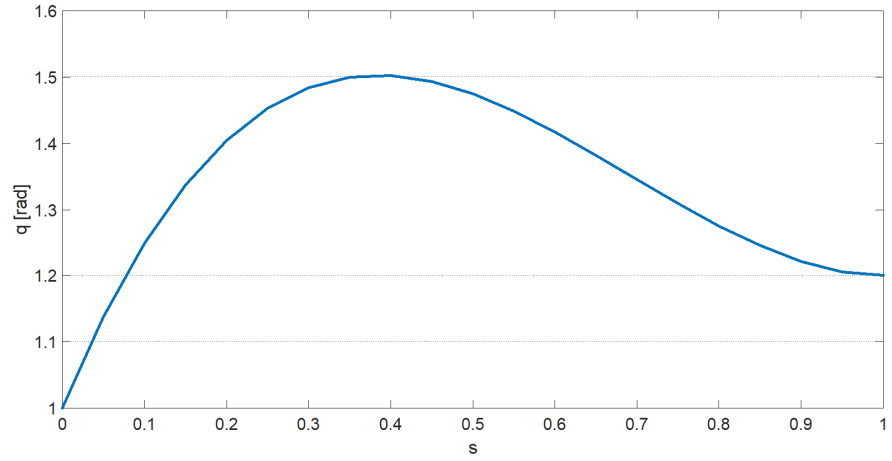


Figure 4.6: A path much longer than needed

Although a 3rd order polynomial function is sufficient for offline optimization, it is not a good solution in a online planning where the catch point could change. For continuity of the robot's motion, when the path is regenerated, the path should be at least $\boldsymbol{C^1}$ at the switching point:

$$\boldsymbol{q_{new}(0) = q_{pre}} \tag{4.69}$$
$$\boldsymbol{q'_{new}(0) = q'_{pre}} \tag{4.70}$$

And the path determined by constraints could be much longer than needed, as shown in Figure 4.6. This problem becomes serious when the robot is close to the catch point or when it moves fast. In many experiments, such a path results in infeasible problem.

A 4th order polynomial is used to remedy this problem. The additional one degree of freedom is used to minimize the difference between previous path and the new path. Let $\boldsymbol{c}$ be the vector of coefficients of the new path, the optimization problem is :

$$\min_{c} \quad \sum_{i=1}^{M} ||\boldsymbol{q_{new}(s_i) - q_{old}(s_i)}||_2^2 \tag{4.71}$$
$$\boldsymbol{s.t.} \quad \boldsymbol{q(1) = q_{catch}} \tag{4.72}$$
$$\frac{\boldsymbol{dq}}{\boldsymbol{dt}}|\boldsymbol{(1) = 0} \tag{4.73}$$
$$\boldsymbol{q_{new}(0) = q_{pre}} \tag{4.74}$$
$$\boldsymbol{q'_{new}(0) = q'_{pre}} \tag{4.75}$$

A sequence of points $\boldsymbol{s_i}$ is used to discretize the original problem. This is a quadratic programming problem (QP). So it could be solved easily.

4.7.2 Trying to find an optimal path

The goal of this project is to plan a thermal energy optimal trajectory for a robot to catch a ball. In a decoupled approach, the choice of path will definitely influence the total thermal energy. It even influences the feasibility of the problem. So is there a way to find a path that prevails other paths according to thermal energy standard? This question seems hard to answer since the idea of the decoupled approach is to separate system dynamics from path planning. We have no time information about the path (it would be called a trajectory if it has time information). But there is a heuristic way to evaluate a path in terms of thermal energy.

The joint torque could be divided into two parts: 1) static part:

$$\boldsymbol{\tau_s} = \boldsymbol{G(q)} \tag{4.76}$$

and 2) dynamic part:

$$\boldsymbol{\tau_d} = \boldsymbol{M(q)\ddot{q} + c(q,\dot{q})\dot{q}} \tag{4.77}$$

The static part is used to overcome the gravity and the dynamic part is used to overcome the inertia and Coriolis force. We don't need any time information to evaluated the static part. The second part can't be determined precisely. But intuitively a smooth path requires less accelerations. Therefore, the following cost function is proposed:

$$\boldsymbol{f(\xi)} = \frac{1}{2}\boldsymbol{w_g}||\boldsymbol{G(\xi)}||_2^2 + \frac{1}{2}\boldsymbol{w_d}||\boldsymbol{K_d\xi}||_2^2 + \frac{1}{2}\boldsymbol{w_a}||\boldsymbol{K_a\xi}||_2^2 \tag{4.78}$$

It subjects to the same constraints mentioned above. $\boldsymbol{\xi}$ is a vector that contains joint coordinates discretized with the joint limits.

$$\boldsymbol{\xi} = [\boldsymbol{q_1^1, q_1^2, ..., q_1^N, q_2^1, ..., q_6^N}]^T \tag{4.79}$$

The first term is the static part of the joint torque. The second and third terms represents the 'velocity' and 'acceleration' of this path. They are velocity and acceleration if the time interval between two points is the same as in real case. $\boldsymbol{K_d}$ and $\boldsymbol{K_a}$ are finite differencing matrix:

$$\boldsymbol{K_d} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{...} & \mathbf{0} \\ \mathbf{-1} & \mathbf{1} & \mathbf{0} & \mathbf{...} & \mathbf{0} \\ \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{...} & \mathbf{0} \\ \mathbf{...} & & & & \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{...} & \mathbf{-1} \end{bmatrix} \otimes \boldsymbol{I_{6\times 6}}$$

and

$$\boldsymbol{K_a} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{...} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{...} & \mathbf{0} \\ \mathbf{-1} & \mathbf{0} & \mathbf{1} & \mathbf{...} & \mathbf{0} \\ \mathbf{...} & & & & \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{...} & \mathbf{-1} \end{bmatrix} \otimes \boldsymbol{I_{6\times 6}}$$

$\otimes \boldsymbol{I_{6\times 6}}$ means repeating the left matrix along the diagonal of $\boldsymbol{I_{6\times 6}}$ for 6 times.

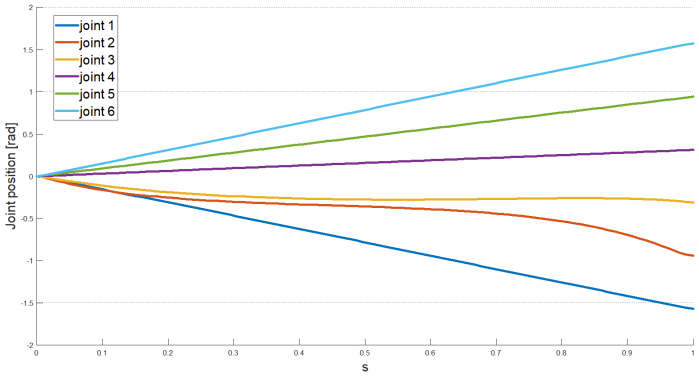


Figure 4.7: Path solved with a large $\boldsymbol{w_d}$ or $\boldsymbol{w_a}$

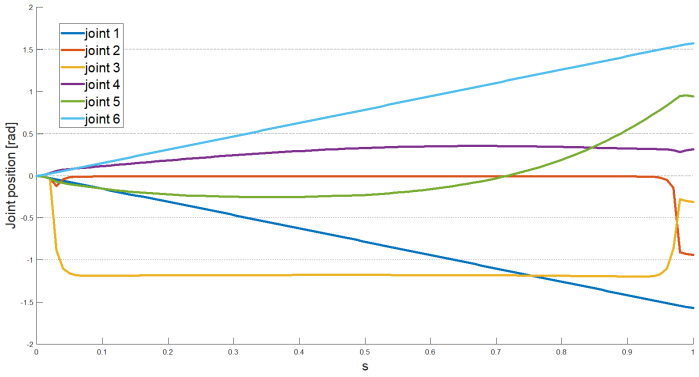


Figure 4.8: Path solved with a larger $\boldsymbol{w_g}$

The weighting factors $\boldsymbol{w_g}, \boldsymbol{w_d}, \boldsymbol{w_a}$ influence the results significantly. If $\boldsymbol{w_d}$ or $\boldsymbol{w_a}$ is large, the result is shown in Figure 4.7. The 2nd joint is the elbow of IRB 120, it is most influenced by the gravity so it tries to remain in vertical position. Other joints just take the shortest path. If $\boldsymbol{w_g}$ is increased, the result is shown in Figure 4.8. Light-loaded joints still go through the shortest path while other joints tries to stay in vertical position. The sharp angle at the beginning and at the end is caused by the 'velocity' constraint. Unlike the continuous path, these constraints can't be just imposed on the first and last points of the path. We need to impose $\boldsymbol{q_2 - q_1 = 0}$ at the beginning and release the constraint gradually.

The disadvantage of using this path generator is the slow computation speed. To have a smooth path, a large $\boldsymbol{N}$ is needed. And the result is a NLP with a dimension of $\boldsymbol{6N}$. For $\boldsymbol{N = 100}$, the time used to solve this NLP is more than 6 seconds. And the velocity constraint makes the situation worse. If the following constraints are imposed on the first five and last five points:

$$\boldsymbol{q_{k+1} - q_k \leq 0.01k} \tag{4.80}$$

the computation time increased to nearly 8 seconds. The ball will fall on the ground before the robot decides which direction to go. Therefore, the 4th order polynomial is used in this project for faster speed.

4.8 Conclusion

By using the decoupled approach, path planning and motion planning are separated. The nonlinear transformation converts the problem into a convex problem so that finding global optimizer is guaranteed. To provide real-time feedback control, the problem is solved repeatedly and a linear interpolation is used to provide control inputs for every sample period. There are two path ways of robot states feedback. One is through system dynamics $\boldsymbol{m}(\boldsymbol{q}), \boldsymbol{c}(\boldsymbol{q}, \boldsymbol{q}'), \boldsymbol{g}(\boldsymbol{q})$ and another is through estimating path parameter $\boldsymbol{s}$. Two estimators, one for continuous path and another for discrete path, are designed for estimating path parameter. To speed up the computation while maintaining accuracy around the catch point, a shrinking horizon is used. And the higher level path planning completes the description of the ball catching scheme.

In the next chapter, simulation results of the proposed scheme is presented and discussed. Some noticeable issues are the effectiveness of energy saving, tracking error and feasibility.

Chapter 5

Simulation Results and Discussion

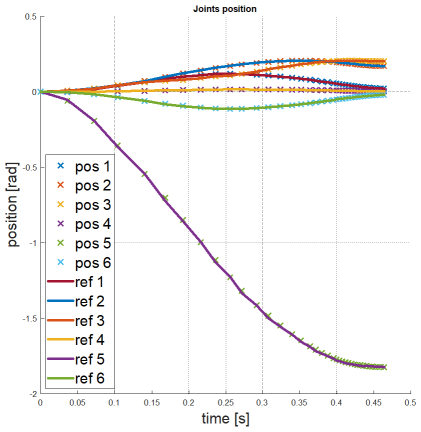
The scheme to solve the ball catching problem has been described in previous chapters. It is time to implement the scheme and see if it will work as expected. MATLAB code used in this chapter is available on GitHub[1]. CasADi [18] is the main tool used in this project. It is a general-purpose tool for numerical optimization problems that require gradient information. The solver for NLPs is IPOPT [21]. Robot model is built using SymPyBotics [16] and Robotics Toolbox [14]. One may find some C++ code on the GitHub page using acados [31] but it is not discussed here. The results are simulated in MATLAB R2017b that runs on a Intel i7-6700HQ processor with 8 GB RAM.

5.1 Results with Changing Catch Point

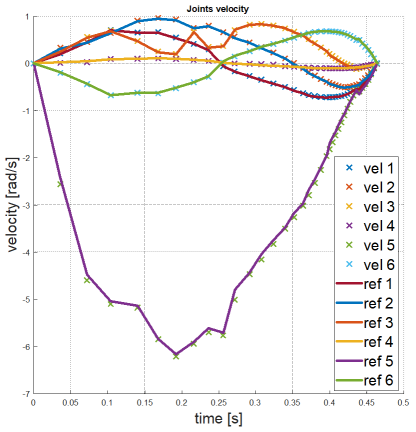
One major motivation to use online optimization is to track a changing path. In this simulation, the ball's initial position and velocity are changed randomly so that the catch point changed for several times. Results are shown in Figure 5.1. The number of collocation points is chosen as $\boldsymbol{N = 10}$.

Changes in catch point, and hence in path, could be seen clearly in the velocity results (Figure 5.1b). Although the reference changes, the robot is still able to track it. Figure 5.1c shows the torque calculated by linear interpolation. And Figure 5.1d is the optimal torque at midpoints of grids. The linear interpolation does its job as expected. The nonlinear $\boldsymbol{\tau}$ is recovered from the optimal $\boldsymbol{a(s)}$ and $\boldsymbol{b(s)}$. We could find discontinuities in joint torque between different time intervals. Three factors are responsible. The first is discontinuous $\boldsymbol{a(s)}$ which is inevitable. But as we will see in next section, if the path remains the same, fluctuation of $\boldsymbol{a}$ will gradually decrease with a shrinking horizon. The second is change in catch point and path. The last one is the singularity in robot dynamics. The shrinking horizon could be clearly seen in these figures. At the end of tracking, the time interval is even shorter than sample period ($\boldsymbol{T_s = 4ms}$). Keep solving the NLP when the time interval is so small is not very meaningful. So the terminal operation is engaged in the last part of the tracking.

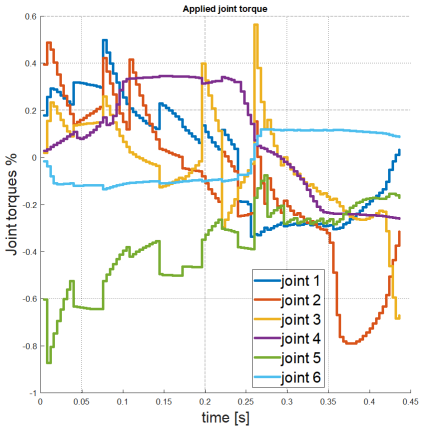
[1] https://github.com/xiyufu/Energy-optimal-ball-catching-robot



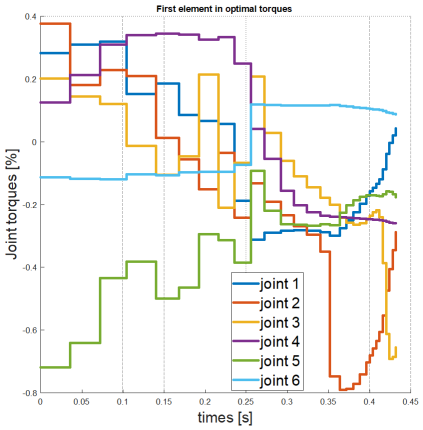
(a) Result:position, changing catch point



(b) Result:velocity, changing catch point



(c) Result:torque from linear interpolation



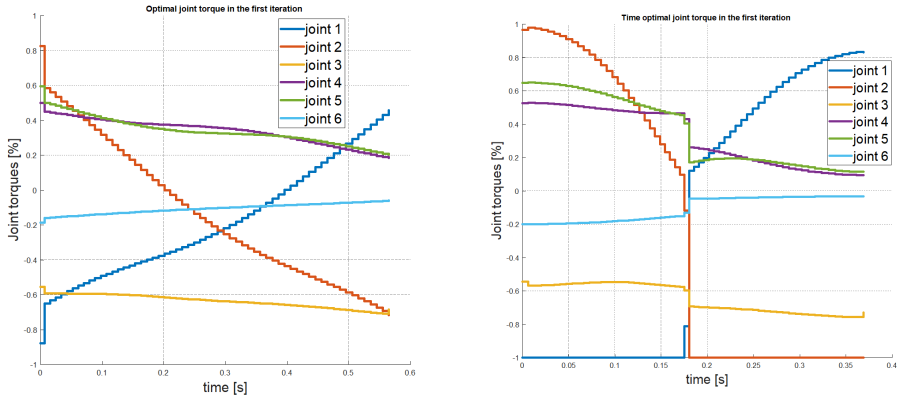
(d) Result:torque from NLP directly

Figure 5.1: Simulation result with a changing catch point

Average time used for solving the optimal control problem (including evaluating $\boldsymbol{m}(\boldsymbol{s}), \boldsymbol{c}(\boldsymbol{s}), \boldsymbol{g}(\boldsymbol{s})$ is 0.036 seconds. And average time used for one linear interpolation is 0.002 seconds. Providing control inputs for every sample period introduces time delay and it might cause instability. One possible remedy is to run path parameter estimation and interpolation in parallel. Solving the optimal control problem needs 9 sample periods. Introducing a linear predictor is necessary if the scheme is to be deployed on a real robot. Another solution is to use more efficient (but less accurate) solvers like acados and implement the scheme in C++.

5.2 Comparison of Thermal Energy

In this section, the thermal energy optimal control is compared with time optimal control. The catch point is fixed from now on so that we could discuss the optimal control problem



(a) Thermal energy optimal joint torques

(b) Time optimal joint torques

Figure 5.2: Comparison between two optimal control problems

itself. The optimal joint torques in both problems are plotted in Figure 5.2.

We could notice that there is always one joint saturated in the time optimal control. There is one switch in the middle of this operation. It means a change in the active set of the NLP. In the energy optimal control, no joints is saturated. The duration of trajectory and thermal energy generated are listed in Table 5.1. The energy optimal control results a slower trajectory with less heat generation as expected. One strange thing in these two

Table 5.1: Comparison of two optimal control

	Time optimal	Energy optimal
Duration [s]	0.376	0.575
Energy (normalized) [s]	0.8216	0.5589

figures is that there is a 'jump' at the beginning. This 'jump' is obviously seen in the energy optimal figure. It is caused by the constraint on the initial value of $\boldsymbol{b}(\boldsymbol{s})$.

$$\boldsymbol{b}(\mathbf{0}) = \boldsymbol{b}_{\boldsymbol{init}} \tag{5.1}$$

This constraint will be further explored in the following sections.

The number of collocation points is increased to $\boldsymbol{N} = \mathbf{60}$. The time optimal control problem is more difficult to solve than the energy optimal control problem and it requires a fine grid. This is the internal disadvantage of the *direct collocation method*. As being introduced in chapter 2, the collocation method transforms the differential equation into algebraic equations at *collocation points*. And the constraints are only imposed on these points. There is no guarantee about the rest part. In time optimal control problem, at least one of the joint is always saturated to have the fastest speed. $\boldsymbol{\tau}$ is piecewise nonlinear within one interval but only the midpoint is constrained. Therefore, the saturated joint

has at least half of the time interval violating torque constraints. When this torque is applied to the robot, the motion of the saturated joint will be slower than expected due to saturation. And after feedback, solving the new NLP will introduce even more constraint violation because it tries to compensate for the error. This is a positive feedback. The NLP becomes infeasible after a few iterations.

Could this problem be solved by using other direct methods like the multiple shooting? I don't think so. Because the ODE is an equation of $\boldsymbol{a}$ and $\boldsymbol{b}$ only. $\boldsymbol{a}$ is piecewise constant so the collocation method in fact gives the exact solution to this ODE. And $\boldsymbol{\tau}$ is determined by a linear combination of these two variables. The only way to solve this problem is by using a fine grid such that the error is small and won't cause infeasible problem.

5.3 Tracking Error

The proposed algorithm could drive the robot along the predefined path until the catch point. Because of the decreasing grid size $\boldsymbol{\Delta}$s, we expect a decreasing tracking error. However, this is not the case, as shown in Figure 5.3

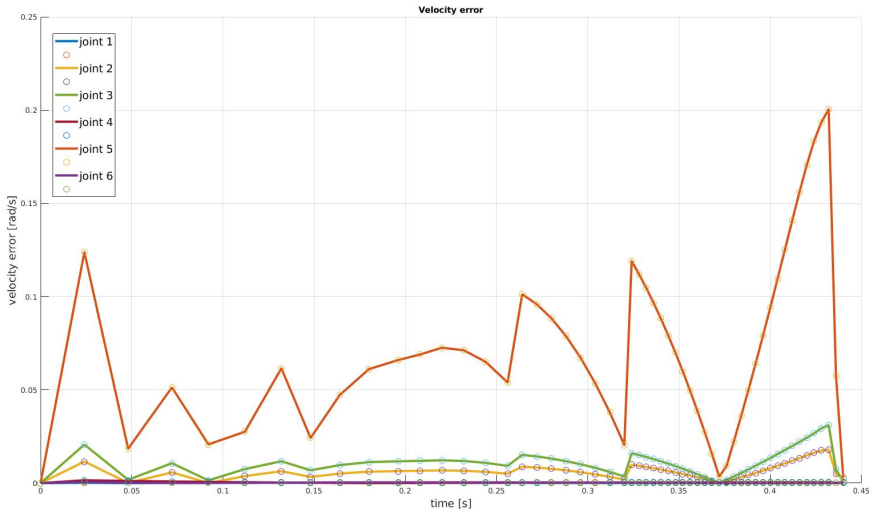


Figure 5.3: Velocity error

Let's take a close look at the error. We first find a large error at the beginning. Then we see a sequence of some 'period', in which the error decrease with smaller and smaller $\boldsymbol{\Delta}$t. But the first error of a period is always a lot larger than the last error of the previous period. After several decreasing periods, we find the error increasing monotonically with $\boldsymbol{\Delta}$t. And finally, the error goes to zero within a few sample period.

This strange behavior is the results of some competing mechanisms that cause errors.

coarse grid: We are using collocation method to transform the optimal control problem whose decision variables have infinite dimensions into an optimization problem with finite dimension decision variables. Therefore, the smaller the grid size, the closer our approximation to the original problem. The result shown in Figure 5.1 uses N = 15. We expect this error will decrease with time because Δs = $(1 - s_{now})/N$ is getting smaller and smaller.

finite sample period: To control the robot with a digital computer, continuous values has to be sampled with a sample period T_s. Because the clock frequency of the computer is not infinite, we have a finite T_s. However, the time interval Δt, within which we apply a constant torque, is usually not an integer multiple of T_s. We always apply the torque a little bit longer or shorter which causes error.

terminating operation: When the robot is very close to the catching point and the grid is so refined that Δt is much smaller than T_s, as shown in the scheme, the algorithm won't try to solve another optimization problem and apply the first optimal torque, but will execute the rest of optimal control history τ_{opt}[i], i = 2,...,N. Because Δt[i] is smaller than T_s, it will combine several adjacent τ_{opt}[i] together and take the average value so that the sum of their Δt[i] is longer than T_s. This average introduces error.

numerical error: There is a relatively large error after the first execution. One parameter, b_{init}, has strong influence on it while b_{init} has no influence on the rest of errors. It's hard to determine what happened here exactly. One reasonable explanation is because of the numerical error introduce by the way we evaluate $\sqrt{b_{k+\frac{1}{2}}}$.

constraint violation: As mentioned in previous section, the collocation method might cause torque constraint violation. Although hasn't been seen in result above, the author did observe its happening in random tests. But the duration is typically short.

The error caused by terminating operation is easy to understand and it only happens at the last few executions (points on the steep decreasing curve). To understand error caused by grid size and simple period, simulation results of different N and T_s are compared. Figure 5.4 shows the velocity error of the 5th joint under three different settings: N=15, T_s=4ms; N=30, T_s=4ms and N=15, T_s=1ms.

By comparing the results of N=15 (orange curve) with N=30 (blue curve), we could find that with a larger N, the error is a little bit smaller. This shows that decreasing the grid size indeed results in a smaller error. However, the influence is not significant. It can also be found that there are less 'period' in the blue curve (N=30). This gives a hint about the origin of the 'period'. For a time interval Δt, the number of samples is determined as $n = \lfloor \Delta t/T_s \rfloor$, $\lfloor x \rfloor$ means round to the greatest integer that is smaller than x. The residual part of Δt decreases within one period. That is the decreasing part in one 'period'. At $\Delta t = n \times T_s$, the error reaches the minimum of this period and after that, number of execution becomes n-1 and the error suddenly jump to a much greater value. And the

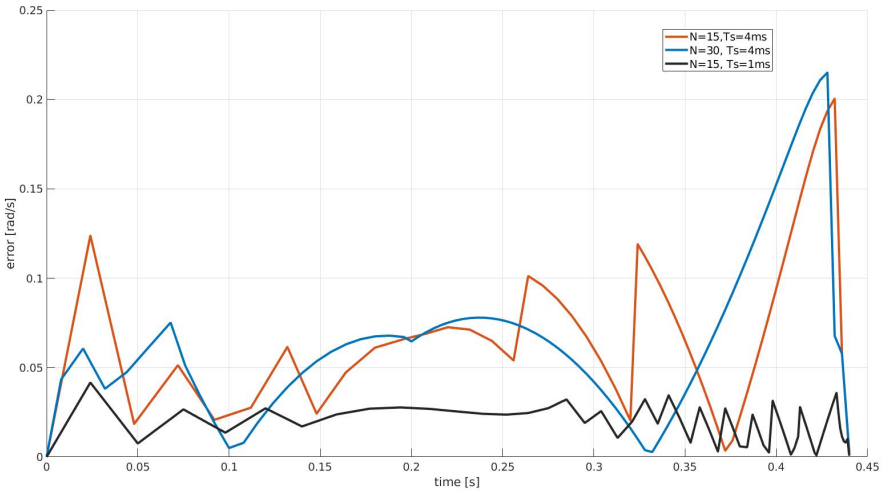


Figure 5.4: velocity errors under different settings

black curve (N=15, $\boldsymbol{T_s = 1ms}$) tells the same story. Because of a smaller sample period, the error curve becomes flat in the middle and the 'period' shows up when changes of $\boldsymbol{\Delta t}$ is comparable with $\boldsymbol{T_s}$.

The monotonically increasing part, whose peak is the largest error, could be explained by the same reason. At this stage, $\boldsymbol{\Delta t[i] < T_s}$, which results in n = 0. But the robot need at least one sample period to execute the control input. So n is round to 1 and the residual error becomes larger and larger while $\boldsymbol{\Delta}$t is getting smaller. Note that for N = 30, this increasing part starts earlier than N = 15. A refined grid leads to larger error. To remedy this problem, the terminating condition is reset as:

$$\boldsymbol{if\ \ (s_{now} < s_{tol})\ ||\ (\Delta t < T_s)} \tag{5.2}$$

$$\boldsymbol{goto:\ terminating\ operation} \tag{5.3}$$

And the result is shown in Figure 5.5. This is better than the old one in Figure 5.3. The blue part in Figure 5.3 is tracking error in terminal operation. It is fluctuating because the reference $\boldsymbol{\dot{q}_{ref}(t)}$ is simply chosen as a straight line.

There is still one strange error in the figure that haven't been explained yet. That is the large error after first execution. By tuning the parameters, it is found that one parameter, $\boldsymbol{b_{init}}$, has strong influence on this error. $\boldsymbol{b_{init}}$ is the initial value of b, there is a constraint in the optimization problem:

$$\boldsymbol{b(0)\ = b_{init}} \tag{5.4}$$

Figure 5.6 shows the resulting error of three different $\boldsymbol{b_{init}: 1 \times 10^{-}6, 1, 10}$. Obviously, the larger $\boldsymbol{b_{init}}$ is, the smaller the first error. However, the rest part of the error curve is not

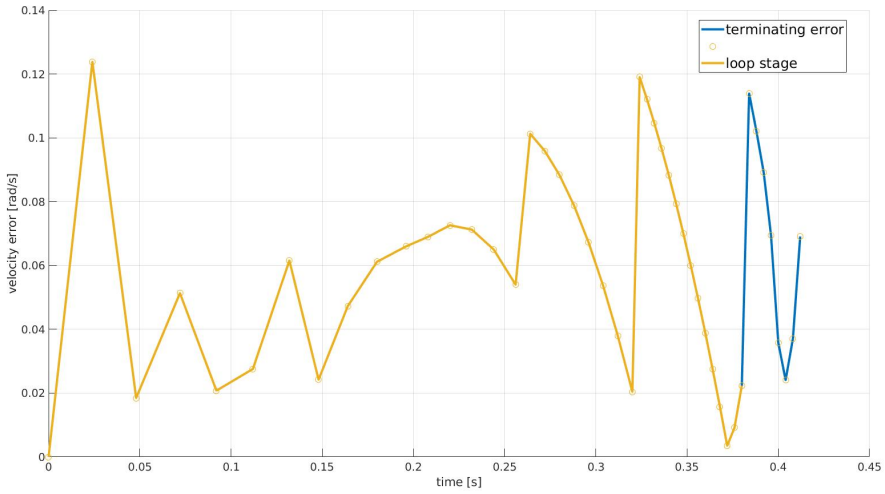


Figure 5.5: Velocity error under new terminating condition

changed. It is hard to say what is happening here exactly. One possible reason is related to how $\sqrt{\boldsymbol{b}_{\boldsymbol{k}+\frac{1}{2}}}$ is evaluated, which is done by the following formula:

$$\sqrt{\boldsymbol{b}_{\boldsymbol{k}+\frac{1}{2}}} = \frac{\sqrt{\boldsymbol{b}_{\boldsymbol{k}}} + \sqrt{\boldsymbol{b}_{\boldsymbol{k+1}}}}{\boldsymbol{2}} \tag{5.5}$$

We know that b is piecewise linear, so we have:

$$\boldsymbol{b}_{\boldsymbol{k}+\frac{1}{2}} = \frac{\boldsymbol{b}_{\boldsymbol{k}} + \boldsymbol{b}_{\boldsymbol{k+1}}}{\boldsymbol{2}} \tag{5.6}$$

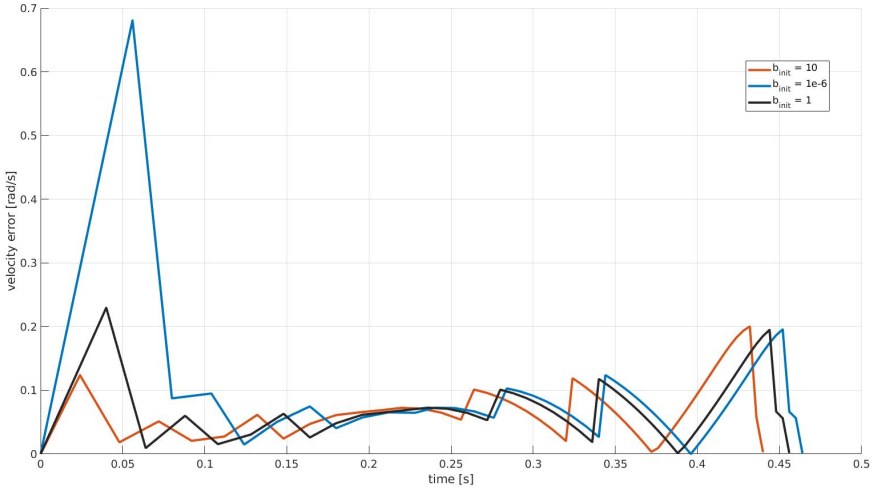


Figure 5.6: error under different $\boldsymbol{b_{init}}$

However, after a nonlinear operation, $\sqrt{b}$, the linearity is lost:

$$\sqrt{b_{k+\frac{1}{2}}} \neq \frac{\sqrt{b_k}+\sqrt{b_{k+1}}}{2} \tag{5.7}$$

If we take a look at the Taylor expansion of $\sqrt{b}$ around b_k:

$$\sqrt{b} = \sqrt{b_k} + \frac{1}{2}\frac{1}{2\sqrt{b_k}}(b-b_k) - \frac{1}{6}\frac{1}{4\sqrt{b_k^3}}(b-b_k)^2 + o((b-b_k)^2) \tag{5.8}$$

It could be seen that the nonlinear part becomes smaller if b_k is larger. And a large b_k results in small error. And how is this numerical error linked to the error of velocity? One reason hides in the way how time intervals are determined from the optimal solutions. They are calculated by the following formula:

$$\Delta t = \frac{\Delta s_k}{\sqrt{b_{k+\frac{1}{2}}}} \tag{5.9}$$

The numerical error caused a smaller estimation of $\sqrt{b_{\frac{1}{2}}}$ (because the square root is a concave function) which will result in a larger Δt. So the first time interval during which we actually applied torques to the robot is longer than expected. And it leads to an overshoot as shown in Figure 5.3 .

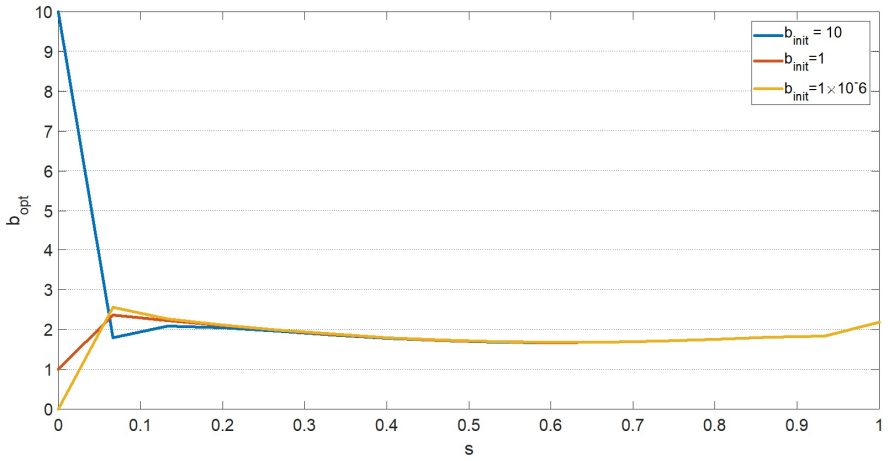


Figure 5.7: b_{opt} with different b_{init}

But why aren't the rest steps influenced by this numerical error? The reason is simple. If we take a look at Figure 5.7 we will find that the change of b is relatively small in the remaining part. Since $\sqrt{b_{\frac{1}{2}}}$ must locates somewhere between $\sqrt{b_k}$ and $\sqrt{b_{k+1}}$, the difference between $\frac{\sqrt{b_k}+\sqrt{b_{k+1}}}{2}$ and $\sqrt{b_{\frac{1}{2}}}$ is also very small when the change of b is small.

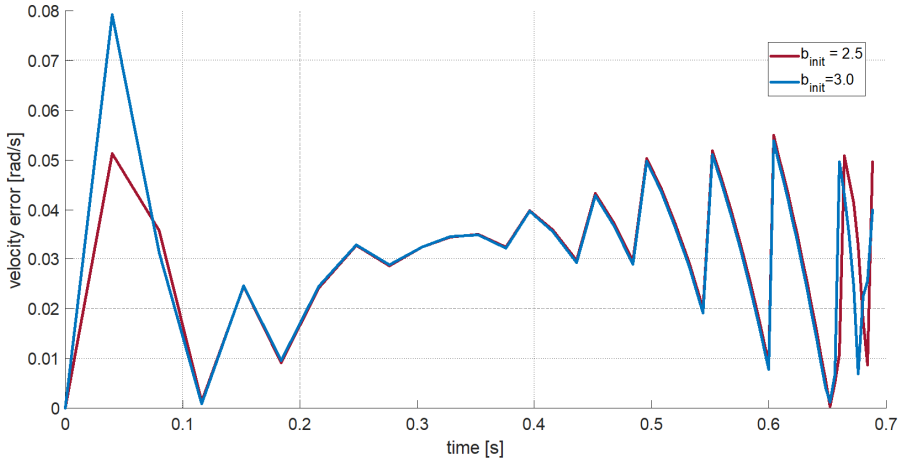


Figure 5.8: Velocity error at b_{init}=2.5 and 3

This argument could be verified by the following result: Set b_{init} to 2.5 and 3. For q_{itc} = [-1.5610;1.5871;-1.4412;1.8197;0.1606;0.6442] and T_{itc} = **2** s, the b_{opt}**(2)** is about 2. Therefore, when b_{init} is set to 2.5, we expect a smaller error than b_{init} = **3** because the difference between b_{init} and b_{opt}**(2)** is smaller. And the result is shown in Figure 5.8. The smaller b_{init} gives smaller error as expected!

5.4 Feasibility

Catching a flying ball is a challenging task even for humans like this author. There are many reasons for the robot not being able to catch the ball. These reasons range from failing to locate the ball to imprecisely estimating velocity of the ball. This section will limit itself to the optimal control problem, that is to assume that all the required inputs are known precisely and, of course, the ball will go inside the robot's workspace.

There are two constraints that decide if there is a solution, namely constraint on catch time and constraints on maximum and minimum torque. Intuitively, when the ball is flying too fast, the robot won't be able to catch it. But which constraint is the limiting factor? For many randomly generated infeasible catch points, both of them are violated at the same time. In the following sections, these constraints will be examined in detail.

5.4.1 Torque constraints

The torque constraints are:

$$\tau_{min} \leq \tau = m(s_{k+\frac{1}{2}})a_k + c(s_{k+\frac{1}{2}})b_{k+\frac{1}{2}} + g(s_{k+\frac{1}{2}}) \leq \tau_{max} \tag{5.10}$$

$$a_k = \frac{b_{k+1} - b_k}{2\Delta s}, \; b_{k+\frac{1}{2}} = \frac{b_{k+1} + b_k}{2} \tag{5.11}$$

For joint i, these constraints could be rewrote as:

$$\frac{\tau_{i,min} - g(s_{k+\frac{1}{2}}) - c(s_{k+\frac{1}{2}})b_{k+\frac{1}{2}}}{m_i(s_{k+\frac{1}{2}})} \leq a_k \leq \frac{\tau_{i,max} - g(s_{k+\frac{1}{2}}) - c(s_{k+\frac{1}{2}})b_{k+\frac{1}{2}}}{m_i(s_{k+\frac{1}{2}})} \tag{5.12}$$

At first glance, one may argue that as long as m_i is greater than zero, we could always find an a_k, such that a_k falls in the range of $[\frac{\tau_{i,min} - g(s_{k+\frac{1}{2}}) - c(s_{k+\frac{1}{2}})b_{k+\frac{1}{2}}}{m_i(s_{k+\frac{1}{2}})}, \frac{\tau_{i,max} - g(s_{k+\frac{1}{2}}) - c(s_{k+\frac{1}{2}})b_{k+\frac{1}{2}}}{m_i(s_{k+\frac{1}{2}})}]$. And the torque constraint will never be violated. However, there is another constraint that prevent us from choose whatever a_k we want, that is:

$$a_k = \frac{b_{k+1} - b_k}{2\Delta s} \tag{5.13}$$

$$b_k \geq 0 \tag{5.14}$$

Put them together, we have:

$$b_{k+1} = b_k + 2\Delta s a_k \geq 0 \tag{5.15}$$

a_k can't be too small ('too negative'), otherwise b_{k+1} would be less than zero. Therefore for some small $\frac{\tau_{i,max} - g(s_{k+\frac{1}{2}}) - c(s_{k+\frac{1}{2}})b_{k+\frac{1}{2}}}{m_i(s_{k+\frac{1}{2}})}$, there is no a_k that is feasible. In other words, to have a feasible problem, the following inequality must hold:

$$\frac{-b_k}{2\Delta s} \leq \frac{\tau_{i,max} - g(s_{k+\frac{1}{2}}) - c(s_{k+\frac{1}{2}})b_{k+\frac{1}{2}}}{m_i(s_{k+\frac{1}{2}})} \tag{5.16}$$

And when would the right part be too small? There are several situations:

1. Since that $c(q(s), q'(s))$ is a quadratic function of $\frac{dq}{ds}$, we could say that if the geometric path has some sharp corners, the problem might be infeasible.

2. At the edge of workspace, the gravity term $g_{k+\frac{1}{2}}$ becomes larger due to larger radius. So the problem is more likely to be infeasible at the edge.

3. Because of the time constraint, there is a lower bound for velocity v and v could be rewritten as $v = \frac{dq}{dt} = \frac{dq}{ds}\frac{ds}{dt} = \frac{dq}{ds}\sqrt{b}$. To meet this requirement, some b_k have to be large enough. When $c(s_{k+\frac{1}{2}})$ is positive, the admissible range of a_k will be smaller due to larger b_k. And the problem is more likely to be infeasible. In other words, the problem becomes infeasible because the remaining time is too short.

Above discussion is based on the assumption that $m(s)_i$ is greater than 0, if it's less than 0, we can exchange the upper and lower limits and draw some similar conclusions.

From Equation 5.15, one may suggest than a smaller $\boldsymbol{\Delta s}$ can reduce the influence of a too negative $\boldsymbol{a_k}$. This is true. For example, if we take a ball whose initial position and velocity are:

$$\boldsymbol{pos_{init}} = [\mathbf{2,0,0}]^T, \ \boldsymbol{vel_{init}} = [\mathbf{-3.15,0,2.86}]^T$$

For N = 15, $\boldsymbol{\Delta s} = \frac{\mathbf{1}}{\boldsymbol{N+1}} = \mathbf{0.0625}$, the problem is infeasible; but for N = 50, $\boldsymbol{\Delta s} = \frac{\mathbf{1}}{\boldsymbol{N+1}} = \mathbf{0.020}$, the problem becomes feasible. Of course, there are limitations on reducing $\boldsymbol{\Delta s}$. The first one is the computation cost while the other one is the sample period.

There is another parameter that also has impact on feasibility. That is $\boldsymbol{b_{init}}$. Take the initial value into the torque constraint, we have:

$$(\frac{\boldsymbol{c_{k+\frac{1}{2}}}}{\mathbf{2}} + \frac{\boldsymbol{m_{k+\frac{1}{2}}}}{\boldsymbol{x\Delta s}})\boldsymbol{b_{k+1}} + (\frac{\boldsymbol{c_{k+\frac{1}{2}}}}{\mathbf{2}} - \frac{\boldsymbol{m_{k+\frac{1}{2}}}}{\boldsymbol{x\Delta s}})\boldsymbol{b_k} \leq \boldsymbol{\tau_{max} - g_{k+\frac{1}{2}}} \tag{5.17}$$

$$\textit{For } \boldsymbol{k=1}: \quad \frac{\boldsymbol{c_{k+\frac{1}{2}}}}{\mathbf{2}} + \frac{\boldsymbol{m_{k+\frac{1}{2}}}}{\boldsymbol{x\Delta s}} = [\mathbf{0.01,1.77,0.69,-0.01,-0.25,0.00}]^T \tag{5.18}$$

$$\frac{\boldsymbol{c_{k+\frac{1}{2}}}}{\mathbf{2}} - \frac{\boldsymbol{m_{k+\frac{1}{2}}}}{\boldsymbol{x\Delta s}} = [\mathbf{-0.01,0.53,0.21,-0.00,-0.08,-0.00}]^T \tag{5.19}$$

$$\boldsymbol{\tau_{max} - g_{k+\frac{1}{2}}} = [\mathbf{75.9,64.6,32.7,6.9,9.0,10.0}]^T \tag{5.20}$$

If $\boldsymbol{b_{init}}$ is too large, we couldn't find a positive $\boldsymbol{b_2}$ that satisfy the torque constraint. As discussed in previous section of error, a large $\boldsymbol{b_{init}}$ is desired for a smaller tracking error. Now it's clear that there is a upper limit of $\boldsymbol{b_{init}}$.

5.4.2 Time constraint

The time constraint is formulated as:

$$\boldsymbol{T} = \sum_{\boldsymbol{k=1}}^{\boldsymbol{N}} \frac{\mathbf{2}\boldsymbol{\Delta s}}{\sqrt{\boldsymbol{b_{k+1}}} + \sqrt{\boldsymbol{b_k}}} \leq \boldsymbol{T_{remain}}$$

Obviously, constraint on time requires shorter path and faster speed. As we have seen in previous section, both shorter path and faster speed are limited by torque constraints. Time and torque constraints form the lower and upper bounds of the feasible set.

And again, $\boldsymbol{b_{init}}$ plays a role in the time constraint. One observation is that larger $\boldsymbol{b_{init}}$ always leads to a shorter total time as shown in Figure 5.6. It is reasonable. By definition, $\boldsymbol{b_{init}} = (\frac{\boldsymbol{ds}}{\boldsymbol{dt}}|_{\boldsymbol{t=0}})^{\mathbf{2}}$ and a larger $\boldsymbol{b_{init}}$ means the robot travels further along the path within a given interval at the beginning. Therefore, it is possible to make a problem feasible by choosing a large $\boldsymbol{b_{init}}$. For example,

$$\boldsymbol{pos_{init}} = [\mathbf{2,0,0}]^T, \ \boldsymbol{vel_{init}} = [\mathbf{-3.15,0,2.86}]^T$$

If $\boldsymbol{b_{init}} = \mathbf{1} \times \mathbf{10^{-}6}$, the problem is infeasible; but for $\boldsymbol{b_{init}} = \mathbf{10}$, the problem becomes feasible. This conclusion is the opposite of the previous chapter! If the torque constraint is the limiting factor, we should lower $\boldsymbol{b_{init}}$ and if the time constraint is the limiting factor, we should elevate it.

Although the objective function is (thermal) energy, the time constraint is usually inactive when the problem has a solution. It is strange at the first glance since our objective requires less torque. It seems that the robot should use up all the remaining time to move slower which means an active time constraint. However, we should notice that there is time in the objective too:

$$E = \sum_{k=1}^{N} \sum_{i=0}^{6} \frac{\tau_i^2(k)}{\tau_{i,max}^2} \frac{2\Delta s}{\sqrt{b_{k+1}} + \sqrt{b_k}} = \sum_{k=1}^{N} \sum_{i=0}^{6} \frac{\tau_i^2(k)}{\tau_{i,max}^2} \Delta t_k \tag{5.21}$$

Therefore, not only a smaller torque but a shorter time could lead to a smaller (thermal) energy. As shown in [32], a very slow motion might increase energy consumption of an industrial robot significantly. One might have such experience in real life: When one tries to do some push-ups, it is always easier to do more if the speed is fast.

5.4.3 A little more about b_{init}

We need a constraint on b_{init} to have a continuous velocity profile in the trajectory. During iterations, this value is inherited from last execution. But for the first iteration, there is a degree of freedom. It is shown in previous sections that a large b_{init} is desired for a smaller error and a faster movement. And the upper limit is set by the torque constraint.

As shown in previous chapter, our optimization problem is convex. There exist a global optimal solution:

$$[a\ b]^T = [a_{opt,1}, a_{opt,2}, \dots, a_{opt,N}, b_{opt,1} = b_{init}, \dots, b_{opt,N+1}]^T \tag{5.22}$$

A natural question is, will the constraint on the first b_{init} influence the global optimal solution? The answer is yes but not much.Take the following example,

$$pos_{init} = [2,0,0]^T,\ vel_{init} = [-1,0,9]^T \tag{5.23}$$

The best b_{init} that gives the smallest energy is about 5.5. If we set it to 10, we get an increase in energy but not much as shown in Table 5.2. We could also take a look at three

Table 5.2: Comparison of thermal energy

b_{init}	5.5	10
E	0.0849	0.0853

different set of b_{opt} as shown in Figure 5.7. We can find they converge to the same value after one or two steps. This result is caused by the convex nature of the NLP.

5.5 Conclusion

The simulation results in this chapter has shown the scheme proposed could control the robot to follow a predefined path in a thermal energy optimal way. It also shows that a time optimal problem is harder to solve due to the property of collocation method. Tracking

error is analyzed and influences of different factors are tested. Feasibility problem is also discussed. The time constraint and torque constraints are the lower and upper bound of feasible set. On the contrary to expectation, the energy optimal trajectory is not necessary the slowest one. A special parameter $\boldsymbol{b_{init}}$ is shown to have impacts on both the tracking error and the feasibility. A larger $\boldsymbol{b_{init}}$ is desired for a smaller tracking error. But if it is too large, the problem becomes infeasible.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, the task of catching a ball optimally is first decomposed into subtasks. These subtasks could be classified into 3 categories, namely perception, planning and control. The perception part includes one high level perception task, estimating the ball's trajectory, and a low level task, determining the corresponding path parameter of current robot states. In a previous work [22], a solution to ball trajectory is proposed. Two estimators are designed in this thesis to solve the low level task. The planning problems are 1) where to catch the ball? and 2) which path to follow? The proposed methods in chapter 3 provide heuristic answers to them. The control problem is solved by two transformations. The first transforms the optimal control problem into path parameter domain and the second transforms the problem into a convex function of $\boldsymbol{a}(\boldsymbol{s})$ and $\boldsymbol{b}(\boldsymbol{s})$. Introducing real-time feedback by online optimization, the robot could catch the ball under unexpected changes. For the coarse grid used in the collocation method, a linear interpolation scheme is introduced to recover nonlinear torque from linear $\boldsymbol{b}(\boldsymbol{s})$ and constant $\boldsymbol{a}(\boldsymbol{s})$.

In simulations, the proposed scheme could track a changing path in an thermal energy optimal way. It is found the number of collocation points not only influences the computation speed but also changes the feasibility of the problem. Because the constraints are only imposed on collocation points, a coarse grid might results in infeasible problems. This problem is serious in time optimal control because at least one joint is always saturated. The way to avoid this problem is by using a finer grid. Different factors about tracking error and feasibility are discussed. The finite sample frequency makes the tracking error decrease and increase periodically. The number of collocation points $\boldsymbol{N}$ does have some impact on tracking error but the influence is not significant. One noticeable factor is $\boldsymbol{b_{init}}$, the initial value of $\boldsymbol{b}(\boldsymbol{s})$. A large $\boldsymbol{b_{init}}$ will not only decrease the tracking error in the first time interval, but can also pull the NLP away from the cliff of time constraint. However, If $\boldsymbol{b_{init}}$ is too large, the NLP will crush on the barrier set by torque constraints.

6.2 Future Work

In the simulation, average computation time used for solving different NLPs is about 0.036 seconds. That is too long for online optimization. Possible remedies are 1) by using more efficient solvers and implementing the scheme in C++; 2) by designing a linear predictor for parameter changes; 3) by running blocks in the scheme in parallel. Implementation of the proposed scheme in C++ using acados is under development. Designing a linear predictor seems promising because the optimal joint torque is linearly dependent on feedback parameters $\boldsymbol{m}(\boldsymbol{s}), \boldsymbol{c}(\boldsymbol{s}), \boldsymbol{g}(\boldsymbol{s})$. But further development of this idea is needed. This author has tried to parallelize the proposed scheme in MATLAB using MATLAB Paralleling Computing Toolbox. However, it seems that CasADi doesn't support parallelization in MATLAB for now. A way to get around this is waiting for exploration.

Another important problem to be solved is the estimation of ball's trajectory. This complex problem involves image processing, 3D extraction and parameter estimation. Finding a solution and implementing it are not trivial tasks in the engineering aspect. One difficulty is how to decrease the estimation sensitivity with respect to image recognition error.

Appendices

Appendix A

Waiting for the Ball

The cost function in the main part of this thesis is:

$$\int_0^1 \sum_{i=1}^{6} \frac{\tau_i^2(s)}{\tau_{max,i}^2} \frac{1}{\sqrt{b(s)}} ds \tag{A.1}$$

This is the thermal energy generated during the movement of the robot. When the robot arrives at the catch point, if there are still some time left before the ball arrives, it has to wait there. And waiting also generates heat. As discussed in chapter 6, the robot usually arrives earlier than the ball so waiting should also be taken into account. The new cost function is:

$$\int_0^1 \sum_{i=1}^{6} (\frac{\tau_i^2(s)}{\tau_{max,i}^2}) \frac{1}{\sqrt{b(s)}} ds + (T_{catch} - T) \sum_{i=1}^{6} \frac{\tau_{i,holding}^2}{\tau_{max,i}^2} \tag{A.2}$$

This function, however, is not convex. The trouble maker is the minus sign in front of T. We can reformulate this function into a new form to have a clear view:

$$\int_0^1 \sum_{i=1}^{6} (\frac{\tau_i^2(s)}{\tau_{max,i}^2} - \frac{\tau_{i,holding}^2}{\tau_{max,i}^2}) \frac{1}{\sqrt{b(s)}} ds + T_{catch} \sum_{i=1}^{6} \frac{\tau_{i,holding}^2}{\tau_{max,i}^2} \tag{A.3}$$

This function is convex in the region $G_1 = \{\tau \mid \sum_{i=1}^{6} (\frac{\tau_i^2(s)}{\tau_{max,i}^2} - \frac{\tau_{i,holding}^2}{\tau_{max,i}^2}) > 0\}$ and concave in the region $G_2 = \{\tau \mid \sum_{i=1}^{6} (\frac{\tau_i^2(s)}{\tau_{max,i}^2} - \frac{\tau_{i,holding}^2}{\tau_{max,i}^2}) < 0\}$. In the entire constraint polytope G it is not clear if A.3 is convex or concave.

During a batch test of 200 randomly generated catch point, there was no example of $\sum_{i=1}^{6} (\frac{\tau_i^2(s)}{\tau_{max,i}^2} - \frac{\tau_{i,holding}^2}{\tau_{max,i}^2}) < 0$. From the batch test, one would wonder whether G_2 is empty for a given path. Intuitively, moving precisely along a path should cost more energy than standing still. However, it's hard to give a proof so this cost function is not used in the project. Using the cost function A.3 results in a trajectory that is a little slower than the old problem's solution. The torque applied to joint 1 in these two problems are plotted in Figure A.1. During the first time interval, the two optimal control inputs look similar. This is because b_{init} is set to a sufficiently large value (10). There is little freedom in choosing $a(1)$ without violating the constraint as mentioned in chapter 6.

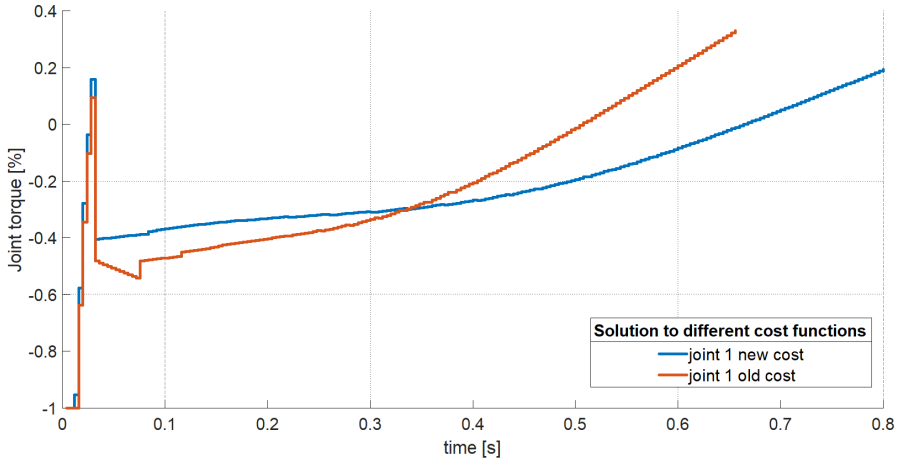


Figure A.1: Optimal control history of joint 1 in different problems

According to the product manual of IRB 120 [2], every motor in it is equipped with a brake. Power consumption during standstill decreases a lot when the brake is engaged. However, the manual only provides the data for joints in zeros position. Modeling and identification are needed before the effect of brakes could be considered.

Bibliography

[1] G. Oriolo, L. Sciavicco, B. Siciliano, and L. Villani, *Robotics: modelling, planning and control*. Springer, 2010.

[2] ABBRobotics, "Irb 120." URL: https://new.abb.com/products/robotics/industrial-robots/irb-120.

[3] M. Brossog, M. Bornschlegl, J. Franke, *et al.*, "Reducing the energy consumption of industrial robots in manufacturing systems," *The International Journal of Advanced Manufacturing Technology*, vol. 78, no. 5-8, pp. 1315–1328, 2015.

[4] R. Saidur, "A review on electrical motors energy use and energy savings," *Renewable and Sustainable Energy Reviews*, vol. 14, no. 3, pp. 877–898, 2010.

[5] M. Vukobratovic and M. Kircanski, "A method for optimal synthesis of manipulation robot trajectories," *Journal of Dynamic Systems, Measurement, and Control*, vol. 104, no. 2, pp. 188–193, 1982.

[6] H. H. Tan and R. B. Potts, "A discrete trajectory planner for robotic arms with six degrees of freedom," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 5, pp. 681–690, 1989.

[7] X. Wang, J. Swevers, J. Stoev, and G. Pinte, "Energy optimal point-to-point motion using model predictive control," in *ASME 2012 5th Annual Dynamic Systems and Control Conference joint with the JSME 2012 11th Motion and Vibration Conference*, pp. 267–273, American Society of Mechanical Engineers, 2012.

[8] B. Hove and J.-J. E. Slotine, "Experiments in robotic catching," *American Control Conference*, 1991.

[9] U. Frese, B. Bauml, S. Haidacher, G. Schreiber, I. Schaefer, M. Hahnle, and G. Hirzinger, "Off-the-shelf vision for a robotic ball catcher," *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*.

[10] M. Riley and C. G. Atkeson, "Robot catching: Towards engaging human-humanoid interaction," *Autonomous Robots*, vol. 12, no. 1, pp. 119–128, 2002.

[11] B. Bäuml, T. Wimböck, and G. Hirzinger, "Kinematically optimal catching a flying ball with a hand-arm-system," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, IEEE, 2010.

[12] R. M. Murray, *Mathematical introduction to robotic manipulation.* Taylor & Francis Ltd, 2017.

[13] B. Du, N. Xi, and E. Nieves, "Industrial robot calibration using a virtual linear constraint.," *International Journal on Smart Sensing & Intelligent Systems*, vol. 5, no. 4, 2012.

[14] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised*, vol. 118. Springer, 2017.

[15] G. G. Slabaugh, "Computing euler angles from a rotation matrix," 1999.

[16] C. D. Sousa, "cdsousa/sympybotics." URL: https://github.com/cdsousa/SymPyBotics.

[17] P. Fisette, B. Raucent, and J. C. Samin, "Minimal dynamic characterization of tree-like multibody systems," *Nonlinear Dynamics*, vol. 9, no. 1-2, pp. 165–84, 1996.

[18] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, In Press, 2018.

[19] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control," in *Fast motions in biomechanics and robotics*, pp. 65–93, Springer, 2006.

[20] R. Bellman, "On the theory of dynamic programming," *Proceedings of the National Academy of Sciences*, vol. 38, no. 8, pp. 716–719, 1952.

[21] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[22] F. Reniers, "Energy-optimal ball catching with a robot using monocular vision," 2017.

[23] C. Smith and H. I. Christensen, "Using cots to construct a high performance robot arm," in *Robotics and Automation, 2007 IEEE International Conference on*, IEEE, 2007.

[24] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward, "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers," *ACM Transactions on Mathematical Software (TOMS)*, vol. 31, no. 3, pp. 363–396, 2005.

[25] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[26] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 489–494, IEEE, 2009.

[27] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[28] K. Shin and N. McKay, "Minimum-time control of robotic manipulators with geometric path constraints," *IEEE Transactions on Automatic Control*, vol. 30, no. 6, pp. 531–541, 1985.

[29] J. E. Bobrow, S. Dubowsky, and J. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The international journal of robotics research*, vol. 4, no. 3, pp. 3–17, 1985.

[30] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2318–2327, 2009.

[31] Acados, "acados/acados." URL: https://github.com/acados/acados.

[32] M. Chemnitz, G. Schreck, and J. Krüger, "Analyzing energy consumption of industrial robots," in *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pp. 1–4, IEEE, 2011.