공학석사 학위논문

에너지 효율 향상을 위한 로봇 머니퓰레이터의 경로 최적화와 스케줄링

Energy-Optimal Trajectory Generation and Task Scheduling for Multiple Robot Manipulators

2016년 8월

서울대학교 대학원

기계항공공학부

최 근 준

ABSTRACT

Energy-Optimal Trajectory Generation and Task Scheduling for Multiple Robot Manipulators

by

Keunjun Choi

School of Mechanical and Aerospace Engineering
Seoul National University

This thesis presents an energy-optimal task scheduling algorithm with a point-to-point trajectory generation method under kinematic and dynamic constraints. Because the energy-optimal trajectory generation is inevitable for performing task scheduling with respect to energy optimality, the integration of them is a big issue in this thesis.

We first propose an energy-optimal trajectory generation algorithm. The optimization problem is defined for multiple waypoints and various boundary conditions with free execution times. The trajectories are parameterized by B-spline curves in the joint space and the objective functions are obtained with joint torques which are calculated by a recursive inverse dynamics method. To

make our algorithm computationally efficient, the gradients for the optimization are calculated analytically. Gaussian quadrature method which is proper for several reasons is used for the integration. We generate the optimal trajectories in several situations to evaluate our algorithm.

We also propose an energy-optimal task scheduling algorithm using dynamic programming method. We first define a problem with four assumptions which can make our problem more practical. Our algorithm determines which robot is optimal for performing each task and finds the optimal time when each task starts and also we optimize the task execution times to minimize the energy consumption. The energy consumption for each task is calculated by energy-optimal trajectory generation algorithm proposed in this thesis. To reduce the computation time of our task scheduling algorithm, we provide an optimal energy consumption measurement which is approximated as a function of the execution time by performing energy-optimal trajectory generation algorithm only four times.

Keywords: Serial open-chain manipulator, energy optimization, point-to-point trajectory planning, base link optimization, task scheduling, B-spline

Student Number: 2014-21857

Contents

Abstract **i**

List of Tables **vi**

List of Figures **vii**

1 Introduction **1**

1.1 Main Contributions of This Thesis 5

1.2 Organization and Preview 6

2 Preliminaries **8**

2.1 Lie Group 8

2.1.1 Special Euclidean Group 8

2.1.2 Generalized Velocity and Force 10

2.1.3 Adjoint Mapping 11

2.2 Kinematics of Serial Open-Chain Manipulators 12

2.2.1 Forward Kinematics 13

2.2.2 Inverse Kinematics 15

2.3 Dynamics of Serial Open-Chain Manipulators 15

2.3.1 Generalized Inertia 16

2.3.2 Dynamics of a Rigid Body 17
2.3.3 Recursive Inverse Dynamics and Its Derivatives 18
2.3.4 Closed Form of Dynamic Equations 20

3 Energy-Optimal Trajectory Generation 22
3.1 Problem Definition 23
3.2 B-Spline Curves 26
3.2.1 B-Splines with Final Time Parameter 30
3.2.2 Partial Differentiation of a B-Spline Curve 32
3.3 Gaussian Quadrature 34
3.4 Algorithm for Generating Energy-Efficient Trajectory 36
3.5 Case Studies 37
3.5.1 Effort 40
3.5.2 Energy Loss 41
3.5.3 Base Link Optimization 41

4 Task Scheduling of Energy-Optimal Trajectories 54
4.1 Problem Definition 55
4.2 Assumptions 57
4.3 Algorithm for Task Scheduling 62
4.3.1 Task Arranging 62
4.3.2 Optimization: Dynamic Programming and Trajectory Generation 63
4.3.3 Cost Function Approximation 64
4.4 Example: Pick and Place Motion 73

5 Conclusion 77

Bibliography 80

국문초록 **84**

List of Tables

1.1 Preceding researches for the energy-optimal point-to-point trajectory generation 1 3

1.2 Preceding researches for the energy-optimal point-to-point trajectory generation 2 4

3.1 Kinemaics and dynamics constraints of an Efort robot 38

3.2 Optimal results for the first case (effort function) without waypoint and payload 45

3.3 Optimal results for the first case (effort function) without with waypoints and payloads 47

3.4 Optimal results for the second case (energy loss function) without waypoint and payload 49

3.5 Optimal results for the second case (energy loss function) without with waypoints and payloads 51

3.6 Results of base link optimization 53

4.1 Result of scheduling optimization 75

List of Figures

2.1 A n-link serial open-chain manipualtor and some symbols for explanation 13
2.2 A rigid body 16

3.1 Problem definition in the joint space 24
3.2 Problem definition in the task space 25
3.3 Function values and compuation times versus the number of control points 39
3.4 Initial, final, and waypoint configuration 44
3.5 Optimal trajectory for the first case (effort function) without waypoint and payload 45
3.6 Position, velocity, acceleration, and torque of each joint for the first case (effort function) with free final time (no waypoint, no payload) 46
3.7 Optimal trajectory for the first case (effort function) with waypoints and payloads 47
3.8 Position, velocity, acceleration, and torque of each joint for the first case (effort function) with free final time (include waypoints and payload) 48

3.9 Optimal trajectory for the second case (energy loss function) without waypoint and payload 49
3.10 Position, velocity, acceleration, and torque of each joint for the second case (energy loss function) with free final time (no waypoint, no payload) 50
3.11 Optimal trajectory for the second case (energy loss function) with waypoints and payloads 51
3.12 Position, velocity, acceleration, and torque of each joint for the second case (energy loss function) with free final time (include waypoints and payload) 52
3.13 Optimal position and orientation of the base link 53

4.1 Task timeline for explaining some parameters 55
4.2 Original schedule and rescheduling 60
4.3 The change of the optimal parameter with respect to the change of the task execution time (the objective function is given by the energy loss function) 69
4.4 Approximated function to optimal energy consumption with respect to task execute time 72
4.5 Simulation setting for pick and place motion 73
4.6 Waypoint conditions or task constraints for pick and place motion 74
4.7 Joint trajectory of each robot for task scheduling 76

1 Introduction

The energy efficiency of industrial applications is emerging as an important issue lately [1]. Although the time optimality is still a major objective for operating factories, the necessity of the energy efficiency has been received and it has become an objective of timely importance. There are mainly three reasons to explain these phenomena.

The first reason is about the environmental changes and increasing the interest in it. In the last part of the 20th century, through the second industrial revolution, the greenhouse gas emissions from the fossil fuel burning have grown explosively, which causes the global warming. After realizing the seriousness, many countries have tried to reduce their greenhouse gas emissions, e.g., in 2012 Europe, the total greenhouse gas emissions were 19.2% below 1990 levels [2]. As the greenhouse gas emissions are directly or indirectly related to the energy consumption, they have made regulations about it [3, 4] and conducted several campaigns for saving the energy, e.g., Eco-point program in Japan [5]. So, in factories which use a lot of energy, they have no choice but to pay attention to reduce the energy consumption.

Second, the cost of electricity in many industrial countries have been steady increasing. It may be related to some social issues or limited natural resources. The important thing is that the operators of factories want to use their energy efficiently for making more profit.

The last reason is that the new generation of the industry is arriving. These days, as the technologies of the Internet and the computer are growing, the new technologies have been developed such as Internet of things, cloud services, and deep learning. They have changed our life styles and customization have been emerging as an important issue in market. These phenomenons and technologies lead to the change of the industry, the new revolution of the industry. In the new generation, the production lines in factories are becoming more complex and the average of the life cycle of the factories is becoming shorter. In addition, the number of robots used in factories has been increasing, as the labor costs have been going up. Therefore the robots which are substitutions for the labors need to be more 'Smart' for performing complex tasks in the new factories and the system for the complex production lines is needed. Here, we can say that, as factories are getting more complex, the energy efficiency is becoming more important. For example, so far the time optimality is more important, however when the production lines are complicated, we can think that the time optimal is not always right. More specifically, when there is a bottleneck in a line, the robots in the parallel lines do not need to do the task as fast as they can, which means that the energy optimality is much better than the time optimality in this example.

When robots are working together for a task, the effect of the energy efficiency can be maximized, for instance, the robots in a production line or an assembly line, welding robots and so on. Because of that, a high-level scheduling with certain purpose, e.g., time optimality or energy optimality, is needed as well as

Table 1.1: Preceding researches for the energy-optimal point-to-point trajectory generation 1

Algorithm	Param. method[1)]	Cost function	Way-points	Kin. const.[2)]	Dyn. const.[3)]
Diken[7]	Sinusoidal	$\int \Sigma_i \lvert\tau_i \dot{q}_i\rvert \mathrm{d}t$	x	pos*	x
Paes et al.[8]	-	$\int \lvert\dot{q}\rvert^2 \mathrm{d}t$	x	pos/vel	o
Martin et al.[9]	B-spline	effort	x	pos	x
Wang et al.[10]	B-spline	effort & payload	x	o	soft constraint
Hansen et al.[1]	B-spline	energy loss	x	o	o

[1)] Parameterization method

[2)] Kinematic constraints: Position & velocity & acceleration constraints

[3)] Dynamic constraint: Torque constraint

* Only position constraint for end-effector

an energy-optimal trajectory generation. In addition, considering all together, integrating the trajectory generation and task scheduling is important [6]. In this thesis, we propose a versatile trajectory generation and the high-level task scheduling algorithm for energy optimality and show how we can integrate the trajectory generation and task scheduling.

To handle those issues, energy-efficiency, the dynamic equations of motion of a $d-$dof (degree-of-freedom) robot is needed. One standard formulation is given as follows:

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

Table 1.2: Preceding researches for the energy-optimal point-to-point trajectory generation 2

Algorithm	Free bound. condition[4)]	Free final time	Analytic gradient	Remarks
Diken[7]	x	x	-	in task space
Paes et al.[8]	x	x	-	identification & opt.
Martin et al.[9]	x	x	o	**
Wang et al.[10]	x	o	o	-
Hansen et al.[1]	x	x	o	-

[4)] Free boundary condition: $\dot{q}$ and $\ddot{q}$ (or V and $\dot{V}$)

** This algorithm can be used for tracking problem

where $q \in \mathbb{R}^d$ is d-dimensional generalized coordinates, $M(q) \in \mathbb{R}^{d \times d}$ is a mass matrix, $C(q, \dot{q}) \in \mathbb{R}^d$ is a Coriolis force, $V(q) \in \mathbb{R}^d$ is a gravity force and $\tau \in \mathbb{R}^d$ is a joint torque. We calculate the joint torques with recursive algorithm [11]. We use recursive algorithm for inverse dynamics and differential inverse dynamic for calculating an objective function and torque constraints. This thesis first addresses a multi-function energy-optimal trajectory generation algorithm, which means that this algorithm can be used for various boundary conditions and multiple waypoints (via-points). Many researches have been dealt with the energy-optimal trajectory generation. Those researches can be sorted into two groups. The first group is for given trajectory [12, 13], and the other is for given points to be passed through (a point-to-point trajectory generation) [1, 7, 8, 9, 10]. In this theses, we focus on the point-to-point trajectory generation. Table 1.1 and Table 1.2 show some preceding

researches and what they can. In addition to Wang's algorithm [10], ours can generate and optimize the trajectory with respect to the energy optimality to be used when there are multiple waypoints or free boundary conditions.

We then address an energy-optimal task scheduling algorithm. It is not easy to generalize task scheduling problems, because there are so many different types of requirements. Almost preceding researches have been considered fixed task execution times and the time optimality. So, in many cases, they solve the problems called *permutation flowshop problem* [14]. However, we cannot use their problem definition and methods for several reasons. For our case, our objective is to reduce the energy consumption. Because the energy consumption depends on the execution time, the execution time is a key optimization parameter for us, which means that performing the energy-optimal trajectory generation is needed for the energy-optimal task scheduling. There is a few researches to handle the energy-optimal task scheduling. Wigström [12] and Vergnano [15] consider the energy-optimal task scheduling with given trajectory. They introduce a scaling factor to stretch or shorten the trajectory and optimize the task execution times for the task scheduling. Our algorithm generates not only the optimal velocity, but also the optimal path and optimize the task execution times with the permutation problem, so it is more challenging.

We now describe in more detail the contributions of this thesis and show how this thesis is organized.

1.1 Main Contributions of This Thesis

The primary challenges of the energy-optimal task scheduling are how to define a problem well for being used in many cases in terms of practical meanings and how to integrate the task scheduling algorithm with the energy-optimal

trajectory generation. Because the performance of the energy-optimal task scheduling is dominated by the performance of the energy-optimal trajectory generation, we should have to propose and implement a computation-efficient and versatile trajectory generation algorithm. Using that energy-optimal trajectory generation algorithm and dynamic programming, we solve the task scheduling problem. This thesis shows the needs and the solution of the integration of the trajectory generation and task scheduling with respect to the energy optimality and emphasizes the appearance of the energy efficiency in complex production lines and factories.

1.2 Organization and Preview

In Chapter 2, we present some notions and algorithms for formulating our problems. We briefly review Lie group theory used in geometric rigid body dynamics, followed by a description of the recursive algorithms for calculating the inverse dynamics and differential inverse dynamics of serial open-chain systems. In addition, we introduce the closed form for dynamic equations of serial open-chain systems which is used in the following chapters.

In Chapter 3, we provide a framework for the energy-optimal trajectory generation which can be used in various cases. We use B-splines for parameterizing trajectories in the joint space. As we suggest B-splines with a normalized time domain, we can handle the time as an optimization parameter. In addition, we use a recursive inverse dynamics algorithm and the Gaussian quadrature method for calculating joint torques and integrating an objective function, respectively. To make our algorithm faster, we exploit analytic gradients of the objective function from the recursive differential inverse dynamics. To evaluate our algorithm, we demonstrate the optimal trajectory generation for Efort robot with several types of the objective functions under position, velocity,

acceleration and torque constraints in various cases, e.g., multiple waypoints, free boundary conditions and free execution times.

In Chapter 4, we propose an energy-optimal task scheduling algorithm. First, we define a problem with several reasonable assumptions which make it more practical. After that, using dynamic programming method, our algorithm optimizes the task execution times and determines which a robot is optimal for performing individual task. Because we have to calculate optimal energy consumption functions by trajectory generation algorithm which is described in Chapter 3 every times, it tasks so many times to get the results of task scheduling. Performing it more efficient, we approximate optimal energy consumption functions using only four optimization results of the trajectory generation algorithm. To evaluate our algorithm, we simulate a pick and place task with two robots.

In Chapter 5, we conclude this thesis with a summary of our results and discuss some directions for future works.

2

Preliminaries

This chapter presents some notions for formulating our problem. We first introduce Lie group theory and then review a recursive inverse dynamics algorithm of serial open chains based on Lie group techniques.

2.1 Lie Group

In this paper, we formulate the equations of motion of a manipulator geometrically with Lie group theory. Before talking about the dynamics of serial open chains, we begin with a brief review of Lie group. A more detailed introduction to this subject can be found in [11, 16, 17, 18].

2.1.1 Special Euclidean Group

Special Orthogonal Group

The special orthogonal group, denoted $SO(3)$, describes the rotation of a rigid body in the three dimensional space and can be defined as the following:

$$SO(3) = \left\{ R \in \mathbb{R}^{3\times 3} | RR^{\mathrm{T}} = R^{\mathrm{T}}R = I, \det(R) = 1 \right\}.$$

The corresponding Lie algebra $so(3)$ consists of 3×3 real skew-symmetric matrices:

$$so(3) = \left\{ s \in \mathbb{R}^{3\times 3} | s^{\mathrm{T}} = -s \right\}.$$

The typical elements of $so(3)$ can be represented as a three dimensional vector $\omega \in \mathbb{R}^3$. Given any three dimensional vector ω, its 3×3 skew-symmetric matrix representation is denoted as

$$[\omega] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}, \quad \text{where } \omega = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix}.$$

Special Euclidean Group

The special Euclidean group, denoted $SE(3)$, represents both the orientation and position of a rigid body in the three dimensional space. It consists of 4×4 real matrices of the form

$$\begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \in SE(3),$$

where $R \in SO(3)$ and $p \in \mathbb{R}^3$. The 0 in the last row denotes a three-dimensional row zero vector. In addition to representing the motion of a rigid body, $SE(3)$, $T \in SE(3) : \{A\} \to \{B\}$, is also used to describe the orientation and position of the coordinate frame $\{B\}$ with respect to the coordinate frame $\{A\}$. The corresponding Lie algebra $se(3)$ has the form

$$\begin{bmatrix} [\omega] & v \\ 0 & 0 \end{bmatrix} \in se(3),$$

where $[\omega] \in so(3)$ and $v \in \mathbb{R}^3$. Like Lie algebra $so(3)$, Lie algebra $se(3)$ also can be represented as a 6-dimensional vector:

$$S = \begin{bmatrix} \omega \\ v \end{bmatrix} \in \mathbb{R}^6, \quad [S] = \begin{bmatrix} [\omega] & v \\ 0 & 0 \end{bmatrix}.$$

The Lie algebra $se(3)$ is related to the corresponding Lie Group $SE(3)$ through exponential mapping. If $S = [\omega, v]^{\mathrm{T}} \in \mathbb{R}^6$ and ω is a unit vector, its Lie group $T \in SE(3)$ can be expressed as follows:

$$\begin{aligned} T &= e^{[S]\theta} \\ &= I + [S]\theta + [S]^2\frac{\theta^2}{2!} + ... \\ &= \begin{bmatrix} e^{[\omega]\theta} & G(\theta)v \\ 0 & 1 \end{bmatrix}, \end{aligned}$$

where θ is scalar, and

$$\begin{aligned} e^{[\omega]\theta} &= I + [\omega]\theta + [\omega]^2\frac{\theta^2}{2!} + [\omega]^3\frac{\theta^3}{3!} + ... \\ &= I + \sin\theta[\omega] + (1 - \cos\theta)[\omega]^2, \\ G(\theta) &= I\theta + (1 - \cos\theta)[\omega] + (\theta - \sin\theta)[\omega]^2. \end{aligned}$$

Explaining its physical meaning, T is a transformation matrix from a coordinate frame $\{A\}$ to a coordinate frame $\{B\}$ with respect to a reference frame $\{0\}$ by rotating about a particular axis at θ and translating in the same direction as the axis with the length $(\omega \cdot v)\theta$. The direction of the above axis is ω and the axis is passing through $\omega \times v$. We can express this relation as follows:

$$T_{0b} = e^{[S]\theta}T_{0a},$$

where T_{0a} and T_{0b} represent the coordinate frame $\{A\}$ and $\{B\}$ with respect to the reference coordinate frame $\{0\}$, respectively.

2.1.2 Generalized Velocity and Force

Let's consider a moving frame whose trajectory with respect to a reference frame is given by

$$T(t) = \begin{bmatrix} R(t) & p(t) \\ 0 & 1 \end{bmatrix} \in SE(3),$$

where $R(t) \in SO(3)$ and $p(t) \in \mathbb{R}^3$. Then, *generalized velocity* can be defined by

$$V = T^{-1}\dot{T} = \begin{bmatrix} [\omega] & v \\ 0 & 1 \end{bmatrix},$$

where $[\omega] = R^{\mathrm{T}}\dot{R}$ and $v = R^{\mathrm{T}}\dot{p}$. The generalized velocity is an element of $se(3)$, and, of course, can be expressed in a 6-dimensional vector:

$$V = \begin{pmatrix} \omega \\ v \end{pmatrix}.$$

The physical meaning of $\omega \in \mathbb{R}^3$ is an angular velocity of the moving frame and $v \in \mathbb{R}^3$ is a linear velocity of the origin of the moving frame, which are expressed in the moving frame. The *generalized force* acting on the body can be defined as

$$F = \begin{pmatrix} m \\ v \end{pmatrix},$$

where $m \in \mathbb{R}^3$ and $f \in \mathbb{R}^3$ represent the moment and force, respectively. The generalized force is known as an element of $dse(3)$, the dual space of $se(3)$, because $F^{\mathrm{T}}V$ is a physically meaningful scalar value.

2.1.3 Adjoint Mapping

Considering $T = (R, p) \in SE(3)$, the adjoint action of T on $V \in se(3)$, $\mathrm{Ad} : SE(3) \times se(3) \to se(3)$, is defined as

$$\mathrm{Ad}_T V = TVT^{-1}.$$

From some calculations, we can find that Ad_T is regarded as a linear transformation, $\mathrm{Ad}_T : se(3) \to se(3)$, which is defined by a 6×6 matrix

$$\mathrm{Ad}_T = \begin{bmatrix} R & 0 \\ [p]\,R & R \end{bmatrix}.$$

The coadjoint action of T on $V^* \in dse(3)$ which is the dual of V, $\mathrm{Ad}_T^* : dse(3) \to dse(3)$, is defined by a 6×6 matrix

$$\mathrm{Ad}_T^* = \mathrm{Ad}_T^{-\mathrm{T}}.$$

The adjoint mapping can be used in a transformation rule. For explaining the transformation rule, let $\{A\}$ and $\{B\}$ be two coordinate frames attached to the same body, and $T_{ab} \in SE(3) : \{A\} \to \{B\}$ represents the orientation and position of the coordinate frame $\{B\}$ with respect to the coordinate frame $\{A\}$. The generalized velocities of $\{A\}$ and $\{B\}$, V_a and V_b, respectively, have the following relation:

$$V_a = \mathrm{Ad}_{T_{ab}} V_b.$$

And also the generalized forces viewed from the body frames $\{A\}$ and $\{B\}$, F_a and F_b, respectively, have the following relation:

$$F_b = \mathrm{Ad}_{T_{ab}}^* F_a.$$

Let's consider $T = e^{[S]\theta}$, where $S = (\omega, v) \in se(3)$ is constant and $\theta = \theta(t)$ is a function of time. The derivative of Ad_T with respect to t can be expressed as follow:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\mathrm{Ad}_T &= \mathrm{ad}_{\frac{\mathrm{d}T}{\mathrm{d}t}T^{-1}}\mathrm{Ad}_T \frac{\mathrm{d}\theta}{\mathrm{d}t} \\ &= \mathrm{ad}_S \mathrm{Ad}_T \frac{\mathrm{d}\theta}{\mathrm{d}t}, \end{aligned}$$

where $\mathrm{ad}_S : se(3) \to se(3)$ defined as

$$\mathrm{ad}_S = \begin{bmatrix} [w] & 0 \\ [v] & [w] \end{bmatrix}.$$

2.2 Kinematics of Serial Open-Chain Manipulators

Finding where the end-effector of a robot is or which joint angles can make the desired posture of the end-effector are basic issues in the robotics field.

In this chapter, we briefly represent how to deal with these issues using Lie group.

2.2.1 Forward Kinematics

The forward kinematics is to find the orientation and position of the end-effector frame from joint angles. To help our explanation, we use a simple model as shown in Figure 2.1. Note that, in this paper, we focus on serial open-chain manipulators and every joint is a revolute joint. Given the serial

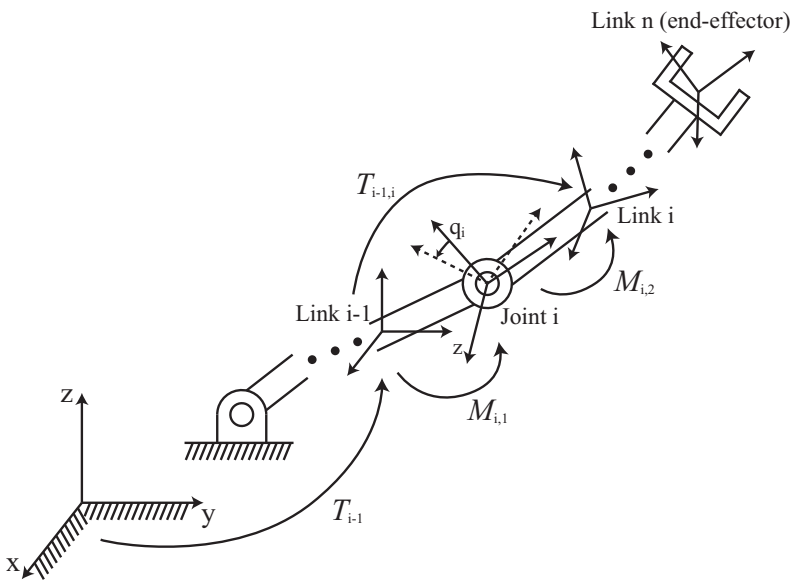


Figure 2.1: A n-link serial open-chain manipulator and some symbols for explanation

open chain manipulator, the following is a list of symbols used in the forward kinematics:

- n = the number of links (exclude ground) or joints
- i = the index of the link or joint
- $q_i \in \mathbb{R}$ = the angle of joint i

- $T_i \in SE(3)$ = the orientation and position of the frame attached to the link i
- $T_{i-1,i} \in SE(3)$ = the orientation and position of the frame attached to the link i with respect to the frame attached to the link $i-1$
- $A_i \in se(3)$ = the screw axis of joint i, if the joint is a revolute joint and the rotation axis is a z-axis of the joint frame, then $A_i = (0,0,1,0,0,0)^{\mathrm{T}}$
- $M_{i,1} \in SE(3)$ = the coordinate transformation from the link $i-1$ frame to the joint i frame
- $M_{i,2} \in SE(3)$ = the cooridnate transformation from the rotated joint i frame to the link i frame

The forward kinematic equation for link i can, therefore, be written as

$$\begin{aligned} T_i &= \prod_{k=1}^{i} T_{k-1,k} \\ &= \prod_{k=1}^{i} M_{k,1} e^{[A_k]q_k} M_{k,2} \end{aligned}$$

So,

$$\begin{aligned} T_n &= M_{1,1} e^{[A_1]q_1} M_{1,2} M_{2,1} e^{[A_2]q_2} M_{2,2} ... M_{n-1,1} e^{[A_{n-1}]q_{n-1}} M_{n-1,2} M_{n,1} e^{[A_n]q_n} M_{n,2} \\ &= e^{[A'_1]q_1} e^{[A'_2]q_2} ... e^{[A'_{n-1}]q_{n-1}} e^{[A'_n]q_n} M, \end{aligned} \tag{2.2.1}$$

where

$$\begin{aligned} \left[A'_i\right] &= M_{1,1} M_{1,2} ... M_{i,1} [A_i] \left(M_{1,1} M_{1,2} ... M_{i,1}\right)^{-1} \\ \Rightarrow A'_i &= \mathrm{Ad}_{M_{1,1} M_{1,2} ... M_{i,1}}(A_i) \\ M &= M_{1,1} M_{1,2} ... M_{n,1} M_{n,2}. \end{aligned}$$

The reason why we reformulate the forward kinematic equation using A'_i, not A_i, in (2.2.1) is to reduce the number of the multiplications. Of course, in doing so, the computation time of the software which includes the forward

kinematics procedure is reduced. Because A_i' and M do not depend on the joint angles, they can be calculated only once at the start point of the software.

2.2.2 Inverse Kinematics

The inverse kinematics is to find the joint angles which make a desired posture of the end-effector. There are several ways to solve this problem. We use a numerical method which is based on Newton-Raphson method. Our problem is to solve $T(q) = \mathcal{T}$, where $T : \mathbb{R}^n \to SE(3)$, $q \in \mathbb{R}^6$ and $\mathcal{T} \in SE(3)$ is a desired orientation and position of the end-effector. Considering a body Jacobian defined as

$$J(q) = \left[\begin{array}{c|c|c|c} \mathrm{Ad}_{\left(e^{[A_2']q_2}\ldots e^{[A_n']q_n}M\right)^{-1}}A_1' & \ldots & \mathrm{Ad}_{\left(e^{[A_n']q_n}M\right)^{-1}}A_{n-1}' & \mathrm{Ad}_{M^{-1}}A_n' \end{array} \right],$$

where S_i is defined as in (2.2.1), the algorithm for the inverse kinematics is as follows:

Algorithm 1 Inverse Kinematics

1: **given** $\mathcal{T} \in SE(3)$, $q_0 \in \mathbb{R}^n$
2: **while** $\|\mathcal{T} - T(q)\| > \epsilon$ **do**
3: $[S] = \log(T^{-1}\mathcal{T})$
4: Solve $J(q)\Delta q = S$ for Δq
5: $q \leftarrow q + \Delta q$
6: **end while**

2.3 Dynamics of Serial Open-Chain Manipulators

This part describes the dynamic equations of a single rigid body and a serial open-chain manipulator. For more detailed information to this subject, one can refer to [16, 17].

2.3.1 Generalized Inertia

The kinetic energy can be calculated by the sum of the kinetic energies of all the mass particles constituting a rigid body:

$$E_k = \int_{\text{vol}} \frac{1}{2}\|v\|^2 \mathrm{d}m.$$

By introducing a coordinate frame attach to the body, the kinetic energy can be reformulated as the following quadratic form by Lie group:

$$E_k = \frac{1}{2} V^{\mathrm{T}} \mathcal{J} V,$$

where $V \in se(3)$ is the generalized velocity and $\mathcal{J} \in \mathbb{R}^{6\times 6}$ is a *generalized inertia*, which represents the mass and the mass distribution with respect to the frame attached to the body. To obtain an explicit form of the generalized form of the body, let $T = (R, p) \in SE(3)$ be the orientation and position of the body frame with respect to a fix reference frame and $r \in \mathbb{R}^3$ be the position of dm from the origin of the body frame with respect to the body frame as shown in Figure 2.2. Using $\|v\|^2 = \|\dot{p} + \dot{R}r\|^2$, we can get

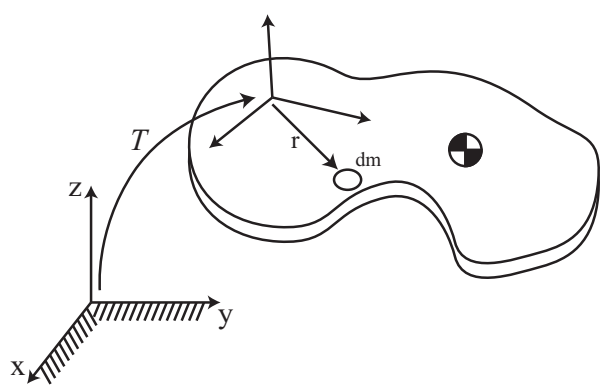


Figure 2.2: A rigid body

$$V = \begin{pmatrix} \omega \\ v \end{pmatrix}, \quad \mathcal{J} = \begin{bmatrix} \int_{vol} [r]^{\mathrm{T}} [r] \, \mathrm{d}m & \int_{vol} [r] \, \mathrm{d}m \\ \int_{vol} [r]^{\mathrm{T}} \, \mathrm{d}m & mI^{3\times 3} \end{bmatrix},$$

where $[\omega] = R^{\mathrm{T}}R$, $v = R^{\mathrm{T}}\dot{p}$, and $I^{3\times3}$ is a 3×3 identity matrix.

Let $\{A\}$ and $\{B\}$ be coordinate frames attached to a rigid body, $\mathcal{J}_a$ and $\mathcal{J}_b$ be the generalized inertias of the body corresponding to the two frames $\{A\}$ and $\{B\}$, respectively. Then, the relationship between two generalized inertias is written as the following:

$$\mathcal{J}_b = \mathrm{Ad}^*_{T_{ab}} \mathcal{J}_a \mathrm{Ad}_{T_{ab}},$$

where $T_{ab} \in SE(3) : \{A\} \to \{B\}$.

2.3.2 Dynamics of a Rigid Body

The *generalized momentum* of a rigid body is defined as

$$L = \mathcal{J}V,$$

where $\mathcal{J}$ and V are the generalized inertia and velocity of the body viewed in a coordinate frame attached to the body, respectively. L is well known as $dse(3)$.

The equations of motion of a rigid body can be written as

$$F = \frac{\mathrm{d}}{\mathrm{d}t} L, \tag{2.3.2}$$

where $F \in dse(3)$ is the sum of the generalized force acting on the rigid body which expressed in the body frame and $L \in dse(3)$ is the generalized momentum of the body. To derivate the generalized momentum, we use the following equation:

$$\frac{\mathrm{d}}{\mathrm{d}t} X = \dot{X} - \mathrm{ad}^*_V X,$$

where $X \in dse(3)$ and $\dot{X}$ is the component-wise time derivative of X and $\mathrm{ad}^*_V : dse(3) \to dse(3)$ is a linear transformation defined as

$$\mathrm{ad}^*_V = \mathrm{ad}^{\mathrm{T}}_V = \begin{bmatrix} [\omega] & 0 \\ [v] & [\omega] \end{bmatrix}^{\mathrm{T}},$$

where $V = (\omega, v) \in se(3)$ is the generalized velocity of the body. Using the above result, the equations of motion of the rigid body (2.3.2) can be written as

$$F = \mathcal{J}\dot{V} - \mathrm{ad}^*_V \mathcal{J} V, \tag{2.3.3}$$

where $\dot{V}$ is the component-wise time derivative of the generalized velocity of the body, V. Note that the dynamic equation of the rigid body is coordinate invariant, which means that this equation can be used in any coordinate frame attached to the body while the generalized force, inertia, velocity, and time derivative of velocity are expressed in the same frame.

2.3.3 Recursive Inverse Dynamics and Its Derivatives

From (2.3.3), we can derive the equations of motion for the link i in Figure 2.1:

$$F_i - \mathrm{Ad}^*_{T^{-1}_{i,i+1}} F_{i+1} = \mathcal{J}_i \dot{V}_i - \mathrm{ad}^*_{V_i} \mathcal{J}_i V_i,$$

where $F_i \in dse(3)$ is the generalized force transmitted to the link i from link $i-1$ through the joint i expressed in the coordinate frame attach to the link i and, of course, V_i, $\dot{V}_i$, and $\mathcal{J}_i$ are the generalized velocity, the component-wise time derivative of V, and the generalized inertia of the link i expressed in the link i frame, respectively. The algorithm for an inverse dynamics can divided into two parts, forward recursions for the forward kinematics and forward differential kinematics, and a backward recursion for calculating generalized forces and torques for all joints. The inverse dynamics can be computed as written in Algorithm 2 (all symbols are defined in Section 2.2.1).

By applying chain rule, the recursive algorithm for the inverse dynamics can be differentiated with respect to an arbitrary scalar $p \in \mathbb{R}$. Algorithm 3 shows the derivative of the recursive inverse dynamics, and it can solve $\frac{\partial \tau}{\partial p}$ with given $\frac{\partial q}{\partial p}$, $\frac{\partial \dot{q}}{\partial p}$, and $\frac{\partial \ddot{q}}{\partial p}$.

Algorithm 2 Recursive Inverse Dynamics

initialization $V_0, \dot{V}_0$

procedure - forward recursion

for $i = 1$ **to** n **do**

$S_i = \mathrm{Ad}_{M_{i,2}^{-1}} A_i$, $M_i = M_{i,1} M_{i,2}$

$T_{i-1,i} = M_i e^{[S_i] q_i}$

$V_i = \mathrm{Ad}_{T_{i-1,i}^{-1}} V_{i-1} + S_i \dot{q}_i$

$\dot{V}_i = \mathrm{Ad}_{T_{i-1,i}^{-1}} \dot{V}_{i-1} + \mathrm{ad}_{V_i} S_i \dot{q}_i + \dot{S}_i \dot{q}_i + S_i \ddot{q}_i$

end for

procedure - backward recursion

for i=n **to** 1 **do**

$F_i = \mathcal{J}_i \dot{V}_i - \mathrm{ad}^*_{V_i} \mathcal{J}_i V_i + \mathrm{Ad}^*_{T_{i,i+1}^{-1}} F_{i+1}$

$\tau_i = S_i^{\mathrm{T}} F_i$

end for

Algorithm 3 Derivative of the Recursive Inverse Dynamics

initialization $\frac{\partial V_0}{\partial p}, \frac{\partial \dot{V}_0}{\partial p}$

procedure - forward recursion

for $i = 1$ **to** n **do**

$\frac{\partial V_i}{\partial p} = \mathrm{Ad}_{T_{i-1,i}^{-1}} \frac{\partial V_{i-1}}{\partial p} + S_i \frac{\partial \dot{q}_i}{\partial p} - \mathrm{ad}_{S_i} V_i \frac{\partial q_i}{\partial p}$

$\frac{\partial \dot{V}_i}{\partial p} = \mathrm{Ad}_{T_{i-1,i}^{-1}} \frac{\partial \dot{V}_{i-1}}{\partial p} + S_i \frac{\partial \ddot{q}_i}{\partial p} - \mathrm{ad}_{S_i} \left\{ \frac{\partial q_i}{\partial p} \mathrm{Ad}_{T_{i-1,i}^{-1}} \dot{V}_{i-1} + \frac{\partial V_i}{\partial p} \dot{q}_i + V_i \frac{\partial \dot{q}_i}{\partial p} \right\}$

end for

procedure - backward recursion

for i=n **to** 1 **do**

$\frac{\partial F_i}{\partial p} = \mathrm{Ad}^*_{T_{i,i+1}^{-1}} \left(\mathrm{ad}^*_{-S_{i+1}} F_{i+1} \frac{\partial q_{i+1}}{\partial p} + \frac{\partial F_{i+1}}{\partial p} \right) + \mathcal{J}_i \frac{\partial \dot{V}_i}{\partial p} - \mathrm{ad}^*_{\frac{\partial V_i}{\partial p}} \mathcal{J}_i V_i - \mathrm{ad}^*_{V_i} \mathcal{J}_i \frac{\partial V_i}{\partial p}$

$\frac{\partial \tau_i}{\partial p} = S_i^{\mathrm{T}} \frac{\partial F_i}{\partial p}$

end for

2.3.4 Closed Form of Dynamic Equations

The previous section shows the recursive algorithm for the inverse dynamics. We easily find that those equation can be expressed as follows:

$$\begin{aligned}
\mathcal{V} &= \Gamma\mathcal{V} + \mathcal{S}\dot{q} + P_0 V_0 \\
\dot{\mathcal{V}} &= \mathcal{S}\ddot{q} + \Gamma\dot{\mathcal{V}} + \mathrm{ad}_{\mathcal{V}}\mathcal{S}\dot{q} + P_0\dot{\mathcal{V}}_0 \\
\mathcal{F} &= \Gamma^{\mathrm{T}}\mathcal{F} + \mathcal{J}\dot{\mathcal{V}} - \mathrm{ad}_{\mathcal{V}}^{\mathrm{T}}\mathcal{J}\mathcal{V} \\
\tau &= \mathcal{S}^{\mathrm{T}}\mathcal{F},
\end{aligned} \tag{2.3.4}$$

where

$$\mathcal{V} = \begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ \vdots \\ V_n \end{pmatrix}, \dot{\mathcal{V}} = \begin{pmatrix} \dot{V}_1 \\ \dot{V}_2 \\ \dot{V}_3 \\ \vdots \\ \dot{V}_n \end{pmatrix}, \mathcal{F} = \begin{pmatrix} F_1 \\ F_2 \\ F_3 \\ \vdots \\ F_n \end{pmatrix}, \dot{q} = \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \vdots \\ \dot{q}_n \end{pmatrix}, \ddot{q} = \begin{pmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \\ \vdots \\ \ddot{q}_n \end{pmatrix},$$

$$P_0 = \begin{pmatrix} \mathrm{Ad}_{T_{0,1}^{-1}} \\ 0^{6\times 6} \\ 0^{6\times 6} \\ \vdots \\ 0^{6\times 6} \end{pmatrix}, \mathcal{S} = \mathrm{diag}\begin{pmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_n \end{pmatrix}, \mathrm{ad}_{\mathcal{V}} = \mathrm{diag}\begin{pmatrix} \mathrm{ad}_{V_1} \\ \mathrm{ad}_{V_2} \\ \mathrm{ad}_{V_3} \\ \vdots \\ \mathrm{ad}_{V_n} \end{pmatrix},$$

$$\Gamma = \begin{bmatrix} 0^{6\times 6} & 0^{6\times 6} & \cdots & 0^{6\times 6} & 0^{6\times 6} \\ \mathrm{Ad}_{T_{1,2}^{-1}} & 0^{6\times 6} & \cdots & 0^{6\times 6} & 0^{6\times 6} \\ 0^{6\times 6} & \mathrm{Ad}_{T_{2,3}^{-1}} & \cdots & 0^{6\times 6} & 0^{6\times 6} \\ \vdots & \vdots & & \vdots & \vdots \\ 0^{6\times 6} & 0^{6\times 6} & \cdots & \mathrm{Ad}_{T_{n-1,n}^{-1}} & 0^{6\times 6} \end{bmatrix}, \mathcal{J} = \mathrm{diag}\begin{pmatrix} \mathcal{J}_1 \\ \mathcal{J}_2 \\ \mathcal{J}_3 \\ \vdots \\ \mathcal{J}_n \end{pmatrix}.$$

Because Γ is a nilpotent matrix, we can get the follow equation:

$$G = (I - \Gamma)^{-1} = \begin{bmatrix} I^{6\times6} & 0^{6\times6} & 0^{6\times6} & \cdots & 0^{6\times6} \\ \mathrm{Ad}_{T_{1,2}^{-1}} & I^{6\times6} & 0^{6\times6} & \cdots & 0^{6\times6} \\ \mathrm{Ad}_{T_{1,3}^{-1}} & \mathrm{Ad}_{T_{2,3}^{-1}} & I^{6\times6} & \cdots & 0^{6\times6} \\ \vdots & \vdots & \vdots & & \vdots \\ \mathrm{Ad}_{T_{1,n}^{-1}} & \mathrm{Ad}_{T_{2,n}^{-1}} & \mathrm{Ad}_{T_{3,n}^{-1}} & \cdots & I^{6\times6} \end{bmatrix}.$$

Using $G = (I - \Gamma)^{-1}$ and $G^{\mathrm{T}} = (I - \Gamma^{\mathrm{T}})^{-1}$, we can reformulate (2.3.4) as

$$\begin{aligned} \mathcal{V} &= G\mathcal{S}\dot{q} + GP_0\mathcal{V}_0 \\ \dot{\mathcal{V}} &= G\mathcal{S}\ddot{q} + G\mathrm{ad}_{\mathcal{V}}\mathcal{S}\dot{q} + GP_0\dot{\mathcal{V}}_0 \\ \mathcal{F} &= G^{\mathrm{T}}\mathcal{J}\dot{\mathcal{V}} - G^{\mathrm{T}}\mathrm{ad}_{\mathcal{V}}^{\mathrm{T}}\mathcal{J}\mathcal{V} \\ \tau &= \mathcal{S}^{\mathrm{T}}\mathcal{F}. \end{aligned} \tag{2.3.5}$$

In our problem, because the base link of a robot which we are focusing on in this thesis is fixed,

$$\mathcal{V}_0 = \begin{pmatrix} 0^{6\times1} \\ \vdots \\ 0^{6\times1} \\ 0^{6\times1} \\ 0^{6\times1} \end{pmatrix}, \quad \dot{\mathcal{V}}_0 = \begin{pmatrix} 0^{6\times1} \\ \vdots \\ 0^{6\times1} \\ 0^{6\times1} \\ G \end{pmatrix},$$

where

$$G \in \mathbb{R}^6 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ g \end{pmatrix}, g = \text{the gravity constant.}$$

3

Energy-Optimal Trajectory Generation

In this chapter, we present a method how to generate energy-optimal point-to-point trajectories using several objective functions and the B-spline parameterization under kinematics and dynamics constraints. During research, what we focus on are how many cases our algorithm can deal with and how fast our algorithm can find the solution. Being useful in terms of practical meanings, the algorithm we suggests should be able to generate the trajectory in as many cases as possible and also as fast as possible. For being versatile, we consider multiple waypoints and various boundary conditions. In addition to that, for the computing efficiency, we calculate gradients of the objective functions analytically for an optimization routine. To verify the performance of our trajectory generation algorithm with respect to the computation time, comparing it to other algorithms is needed. However, unfortunately, the direct comparison is impossible, because the algorithms are not simple and the computation time depends on so many things, e.g., a optimization tool, a coding

style, a memory-managing ability, a hardware spec. Therefore we just figure out how much faster our algorithm with than without analytic gradients.

3.1 Problem Definition

Let $q \in \mathbb{R}^d$ be the joint angles for a serial open-chain manipulator. The equation of motion for our manipulator can be written as [16]:

$$M(q)\ddot{q} + C(q, \dot{q}) + V(q) = \tau, \tag{3.1.1}$$

where $M(q) \in \mathbb{R}^{d \times d}$ is the mass matrix, $C(q, \dot{q}) \in \mathbb{R}^d$ is the Coriolis term, and $V(q) \in \mathbb{R}^d$ includes gravity term and other forces which act at the joints. Although we can get analytic expressions for $M(q)$, $C(q, \dot{q})$, and $V(q)$ of a serial open-chain manipulator [19] and calculate τ using them, we exploit the recursive inverse dynamics method mentioned in previous section. With this system, we want to find the joint trajectory which is minimizing the cost function of the form

$$J(\tau) = \int_{t_i}^{t_f} E(\tau(t))\mathrm{d}t \tag{3.1.2}$$

subject to (3.1.1)

$$\tau(t) = \tau(q(t), \dot{q}(t), \ddot{q}(t))$$

and the joint position, velocity, acceleration, and torque limit are given as

$$q_{min} \leq q(t) \leq q_{max}, \quad \dot{q}_{min} \leq \dot{q}(t) \leq \dot{q}_{max},$$
$$\ddot{q}_{min} \leq \ddot{q}(t) \leq \ddot{q}_{max}, \quad \tau_{min} \leq \tau(t) \leq \tau_{max},$$

where t_i and t_f are the start and end time and $E(\tau(t))$ is a function which represents an instant energy consumption at time t. There are various ways for representing the energy consumption. For our examples, we use an effort

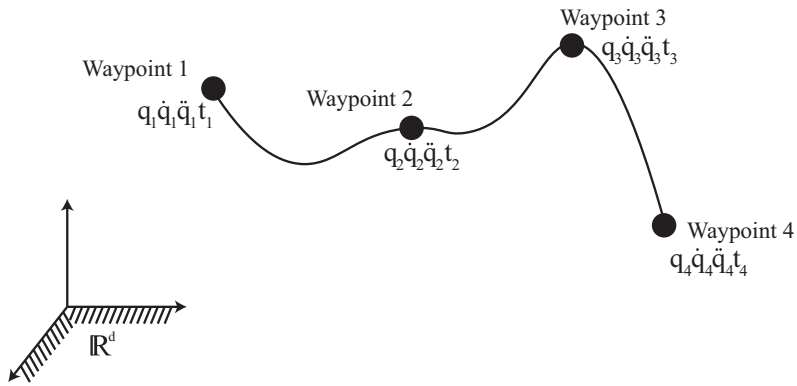


Figure 3.1: Problem definition in the joint space

and an energy loss which are defined as follows:

$$E_{\text{effort}}(\tau(t)) = \tau(t)^{\mathrm{T}}\tau(t)$$

$$E_{\text{energyloss}}(\tau(t)) = \sum_{j=1}^{d} \max\left(i_j(t)v_j(t), 0\right),$$

where i_j and v_j are the current and the voltage of motor in the joint j, respectively:

$$i_j(t) = \frac{1}{k_{m,j}}\left(\frac{\tau_j(t)}{G_j} + J_{m,j}G_j\ddot{q}_j(t)\right)$$

$$v_j(t) = r_j i_j(t) + \ell_j\frac{\mathrm{d}i_j(t)}{\mathrm{d}t} + k_{g,j}G_j\dot{q}_j(t),$$

with a stator resistance r_j, a stator inductance ℓ_j, a motor constant $k_{m,j}$, a back EMF constant $k_{g,j}$ (an electromotive force), a gear ratio G_j, and a motor inertia $J_{m,j}$ of the motor in joint j [20]. Minimizing the effort function is to reduce the exerted joint torques during the execution of a trajectory. Meanwhile, the energy loss function represents an electrical motor power with resistive and inductive energy losses. We can easily guess that the results of the optimization using two objective functions are different. Back to our problem definition, we can include the conditions (q, $\dot{q}$, $\ddot{q}$, t) at each waypoint in joint space, see Figure 3.1, which means that the robot motion must pass

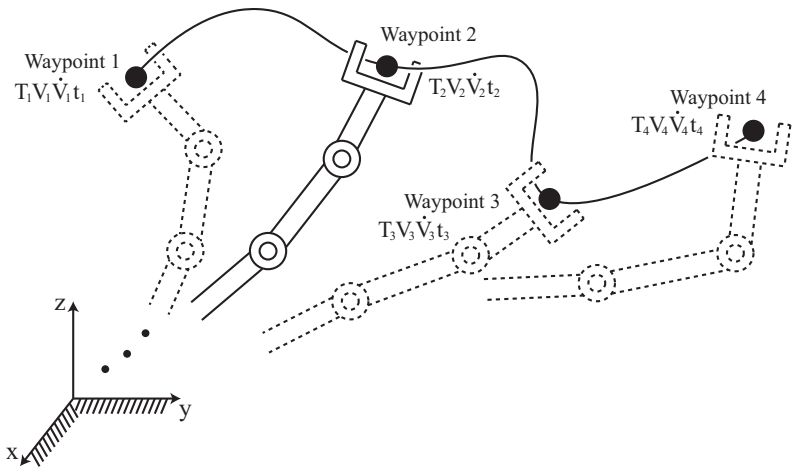


Figure 3.2: Problem definition in the task space

through joint position q at time t with the joint velocity $\dot{q}$ and the joint acceleration $\ddot{q}$. Each condition may be either given or free ('free' means that corresponding variables will be optimized). For example, if waypoint passing time is free, the robot motion pass through the joint position q without any timing constraint. Of course, the condition which is free will be optimized to minimize the cost function. As you can see Figure 3.1, the number of waypoints may be more than two. Moreover, the waypoint conditions may be given in task space; the orientation and position $T \in SE(3)$, spatial velocity $V \in se(3)$, spatial acceleration $\dot{V} \in se(3)$ of the manipulator's end-effector, see Figure 3.2. Like conditions in the joint space, each condition may be either given or free. However, if spatial velocity V is free, T must be given. And if spatial acceleration $\dot{V}$ is given, T and V must be given. With such conditions expressed in task space, we can calculate corresponding conditions in the joint space using an inverse kinematics and some relations:

$$J(q)\dot{q} = V$$

$$\dot{J}(q,\dot{q})q + J(q)\ddot{q} = \dot{V}.$$

If the number of degrees of freedom exceeds the dimension of the task space, inverse kinematics solutions are not unique. Besides, it is possible that inverse kinematics solution does not exists. However, in our problem, we assume that there is at least one solution for inverse kinematics.

A waypoint condition is to represent that an optimal trajectory we find should be passing through the waypoint. There are several ways to add the waypoint conditions to the optimal control problem such as equality condition form, inequality condition form, etc. In this paper, to add the waypoint conditions, dividing the trajectory by the waypoints into several trajectories called sectors, we can reformulate (3.1.2) into

$$\begin{aligned} J(\tau) &= \sum_{i=1}^{W-1} \int_{t_i}^{t_{i+1}} E(\tau(q(t), \dot{q}(t), \ddot{q}(t)))\mathrm{d}t \\ &= \int_{t_1}^{t_2} E(\tau(q(t), \dot{q}(t), \ddot{q}(t)))\mathrm{d}t + ... + \int_{t_{W-1}}^{t_W} E(\tau(q(t), \dot{q}(t), \ddot{q}(t)))\mathrm{d}t \end{aligned} \tag{3.1.3}$$

where W is the number of waypoints, which makes it for us to easily add the waypoint conditions to the optimization problem.

3.2 B-Spline Curves

In order to solve the optimal control problem with a direct method, we need to parameterize a joint trajectory and convert this problem into a parameter optimization problem. For parameterizing the optimal trajectory of each sector with a time variable in the configuration space, a B-spline curve is used. We first introduce about the basic knowledge of B-spline curves. B-spline curves have been widely used in many researches [1, 9, 10, 21]. More detailed information can be found in [22].

When using B-spline curve of order p, the joint trajectory, $q \in \mathbb{R}^d$ is written

as

$$q(t) = q(t; u_{1:n}, c_{1:m}) = \sum_{i=1}^{m} c_i B_{i,p}(t), \quad t \in [0, T] \tag{3.2.4}$$

$$u_\cdot \in \mathbb{R}, c_\cdot \in \mathbb{R}^d, p = n - m, \tag{3.2.5}$$

where $u_\cdot$ and $c_\cdot$ are respectively knots and control points which characterize B-spline curve, t is a time variable, and T is a final time (or traveling time, $T = t_{s+1} - t_s$ for the sector s). The symbols n and m are the number of knots and control points, respectively, and d is the degrees of freedom of a manipulator. $B_{i,p}$ is a B-spline basis function with C^{p-2} continuity which can be derived by means of a recursion formula as follows [10]:

$$\begin{aligned} B_{i,p}(t) &= \frac{t - u_i}{u_{i+p-1} - u_i} B_{i,p-1}(t) + \frac{u_{i+p} - t}{u_{i+p} - u_{i+1}} B_{i+1,p-1}(t), \\ B_{i,1}(t) &= \begin{cases} 1 & \text{if } u_i \le t < u_{i+1} \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{3.2.6}$$

Another method to calculate bases of a B-spline curve is *De Boor's Algorithm* [22].

The derivative of q can be obtained by

$$\frac{\mathrm{d}}{\mathrm{d}t} q(t) = \sum_{i=0}^{m} c'_i B_{i+1,p-1}(t), \tag{3.2.7}$$

where c'_i is defined as follows:

$$c'_i = \frac{p-1}{u_{i+p} - u_{i+1}} (c_{i+1} - c_i), \tag{3.2.8}$$

where c_0 (for $i = 0$) and c_{m+1} (for $i = m$) are defined as zero. Because the derivatives of a B-spline curve is another B-spline curve of order $p - 1$ on the same knots with new $m - 1$ control points $c'_\cdot$, the higher derivative of q can be calculated consequently following a similar procedure.

B-spline has the convex hull property defined by

$$B_{i,p} \geq 0 \quad \text{and} \quad \sum_{i=0}^{m} B_{i,p}(t) = 1.$$

We see that

$$\begin{aligned} q(t) = \sum_{i=1}^{m} c_i B_{i,p}(t) \leq \sum_{i=1}^{m} \bar{c} B_{i,p}(t) = \bar{c} \\ q(t) = \sum_{i=1}^{m} c_i B_{i,p}(t) \geq \sum_{i=1}^{m} \underline{c} B_{i,p}(t) = \underline{c}, \end{aligned} \tag{3.2.9}$$

where $\bar{c}$ is the maximum value of c_i and $\underline{c}$ is the minimum value of c_i. B-spline based parameterization has been widely used in [10, 11].

To set the boundary conditions ($q(0)$, $\dot{q}(0)$, $\ddot{q}(0)$, $q(T)$, $\dot{q}(T)$, $\ddot{q}(T)$) of a B-spline manually, we make the knots like as follows:

$$u_i = \begin{cases} 0 & \text{if } i = 1, ..., p \\ T\Delta(i-p) & \text{if } i = p+1, ..., n-p \\ T & \text{if } i = n-p+1, ..., n \end{cases}, \quad \text{where } \Delta = \frac{1}{n-2p+1}. \tag{3.2.10}$$

By some calculations, we can easily find that, when we make a B-spline curve using above knots, the start point $q(0)$ and the end point $q(T)$ are equal to c_1 and c_m, respectively. B-spline which have the same p knot values on the both sides like above is called *clamped B-spline*. From (3.2.7), we know that the derivative of q is also B-spline curve. The knots u' and the control points c' of $\dot{q}$ can be obtained by

$$\begin{aligned} u'_i &= u_{i+1}, & i &= 1, ..., n-2 \\ c'_j &= \frac{p-1}{u_{j+p} - u_{j+1}}(c_{j+1} - c_j), & j &= 1, ..., m-1, \end{aligned}$$

and we can arrange above equation in matrix form:

$$\begin{bmatrix} c'_1 \\ \vdots \\ c'_{m-1} \end{bmatrix} = \begin{bmatrix} -\frac{p-1}{u_{p+1}-u_2} & \frac{p-1}{u_{p+1}-u_2} & & \\ & \ddots & \ddots & \\ & & -\frac{p-1}{u_{p+m-1}-u_m} & \frac{p-1}{u_{p+m-1}-u_m} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{m-1} \\ c_m \end{bmatrix} \tag{3.2.11}$$

Therefore the number of knots and control points are $n-2$ and $m-1$, respectively, and the order of B-spline $\dot{q}$ is $p-1$. Obviously, B-spline $\dot{q}$ is also clamped B-spline and we can see that $\dot{q}(0)$ and $\dot{q}(T)$ are c'_1 and c'_{m-1}, respectively:

$$\begin{aligned} \dot{q}(0) &= c'_1 = \frac{p-1}{T\Delta}(c_2 - c_1) \\ \dot{q}(T) &= c'_{m-1} = \frac{p-1}{T\Delta}(c_m - c_{m-1}). \end{aligned}$$

With a similar procedure, $\ddot{q}(0)$ and $\ddot{q}(T)$ can be formulated with c_i. Therefore, from those simultaneous equations, we can find that the control points of B-spline $q(t)$ which satisfies the boundary conditions ($q(0)$, $\dot{q}(0)$, $\ddot{q}(0)$, $q(T)$, $\dot{q}(T)$, $\ddot{q}(T)$) subject to following equations:

$$\begin{aligned} c_1 &= q(0) \\ c_2 &= q(0) + \frac{T\Delta}{p-1}\dot{q}(0) \\ c_3 &= q(0) + \frac{3T\Delta}{p-1}\dot{q}(0) + \frac{2T^2\Delta^2}{(p-1)(p-2)}\ddot{q}(0) \\ c_{m-2} &= q(T) - \frac{3T\Delta}{p-1}\dot{q}(T) + \frac{2T^2\Delta^2}{(p-1)(p-2)}\ddot{q}(T) \\ c_{m-1} &= q(T) - \frac{T\Delta}{p-1}\dot{q}(T) \\ c_m &= q(T). \end{aligned} \tag{3.2.12}$$

By the definition of B-spline, we can split B-spline into two B-splines with knots sharing.

$$\begin{aligned} q(t; u_{1:n}, c_{1:m}) &= q_1(t; u_{1:n}, c_{1,1:m}) + q_2(t; u_{1:n}, c_{2,1:m}) \\ &= \sum_{i=1}^{m} c_{1,i} B_{i,p}(t) + \sum_{i=1}^{m} c_{2,i} B_{i,p}(t), \end{aligned} \tag{3.2.13}$$

, where

$$c_{1,i} = \begin{cases} c_i & \text{if } i = 1, 2, 3, m-2, m-1, m \\ 0 & \text{otherwise} \end{cases}$$

$$c_{2,i} = \begin{cases} 0 & \text{if } i = 1, 2, 3, m-2, m-1, m \\ c_i & \text{otherwise.} \end{cases}$$

Using (3.2.12) and (3.2.13), when boundary conditions and T are given, we can easily find that q_1 just depends on the time variable t.

3.2.1 B-Splines with Final Time Parameter

To use the final time T as an optimization parameter, we need to introduce new variable and normalize the domain of B-splines into range of $[0, 1]$. Considering new normalized time-domain variable $\tilde{t}$, the relation between $\tilde{t}$ and t is

$$\tilde{t} = \frac{t}{T}, \quad \tilde{t} \in [0, 1]. \tag{3.2.14}$$

When $q(t)$ is defined by (3.2.4), we define new B-spline in range of $[0, 1]$ with new variable $\tilde{t}$:

$$\tilde{q}(\tilde{t}; \tilde{u}_{1:n}, c_{1:m}) = \sum_{i=1}^{m} c_i \tilde{B}_{i,p}(\tilde{t}), \tag{3.2.15}$$

where

$$\tilde{u}_i = \frac{u_i}{T}, \quad i = 1, ..., n$$
$$= \begin{cases} 0 & \text{if } i = 1, ..., p \\ \Delta(i-p) & \text{if } i = p+1, ..., n-p \\ 1 & \text{if } i = n-p+1, ..., n \end{cases}, \quad \text{where } \Delta = \frac{1}{n-2p+1}.$$

From (3.2.6),

$$\begin{aligned} \tilde{B}_{i,1}(\tilde{t}(t)) &= B_{i,1}(t), \\ \tilde{B}_{i,p}(\tilde{t}(t))) &= \frac{\tilde{t}(t) - \tilde{u}_i}{\tilde{u}_{i+p-1} - \tilde{u}_i}\tilde{B}_{i,p-1}(\tilde{t}(t)) + \frac{\tilde{u}_{i+p} - \tilde{t}(t)}{\tilde{u}_{i+p} - \tilde{u}_{i+1}}\tilde{B}_{i+1,p-1}(\tilde{t}(t)) \\ &= \frac{t/T - u_i/T}{u_{i+p-1}/T - u_i/T}\tilde{B}_{i,p-1}(\tilde{t}(t)) + \frac{u_{i+p}/T - t/T}{u_{i+p}/T - u_{i+1}/T}\tilde{B}_{i+1,p-1}(\tilde{t}(t)) \\ &= \frac{t - u_i}{u_{i+p-1} - u_i}\tilde{B}_{i,p-1}(\tilde{t}(t)) + \frac{u_{i+p} - t}{u_{i+p} - u_{i+1}}\tilde{B}_{i+1,p-1}(\tilde{t}(t)) \\ &= B_{i,p}(t). \end{aligned}$$

Therefore, referring to (3.2.15) it implies that

$$\begin{aligned} q(t; u_{1:n}, c_{1:m}) &= \sum_{i=1}^{m} c_i B_{i,p}(t) \\ &= \sum_{i=1}^{m} c_i \tilde{B}_{i,p}(\tilde{t}(t)) \\ &= \tilde{q}(\tilde{t}(t); \tilde{u}_{1:n}, c_{1:m}). \end{aligned} \tag{3.2.16}$$

Now, let's take derivatives of above equation, then we get

$$\begin{aligned} \dot{q}(t) &= \frac{\mathrm{d}\tilde{q}}{\mathrm{d}\tilde{t}}(\tilde{t}(t))\frac{\mathrm{d}\tilde{t}}{\mathrm{d}t} \\ &= \frac{1}{T}\frac{\mathrm{d}\tilde{q}}{\mathrm{d}\tilde{t}}(\tilde{t}(t)) \\ \ddot{q}(t) &= \frac{1}{T^2}\frac{\mathrm{d}^2\tilde{q}}{\mathrm{d}\tilde{t}^2}(\tilde{t}(t)). \end{aligned} \tag{3.2.17}$$

Using above equations, we can rearrange our problem (3.1.3) into

$$\begin{aligned} J_s(\tau) &= \int_0^T E(\tau(q(t), \dot{q}(t), \ddot{q}(t)))\mathrm{d}t \\ &= \int_0^T E\left(\tau\left(\tilde{q}(\tilde{t}(t)), \frac{1}{T}\frac{\mathrm{d}\tilde{q}}{\mathrm{d}\tilde{t}}(\tilde{t}(t))\frac{1}{T^2}\frac{\mathrm{d}^2\tilde{q}}{\mathrm{d}\tilde{t}^2}(\tilde{t}(t))\right)\right)\mathrm{d}t \\ &= \int_0^T \tilde{E}(\tilde{t}(t))\mathrm{d}t, \end{aligned} \tag{3.2.18}$$

where J_s is an objective function of sector s. Normalizing the domain of B-splines make it possible for us to easily take partial derivatives of $q(t)$, $\dot{q}(t)$, and $\ddot{q}(t)$ with respect to final time T using $\tilde{q}(\tilde{t})$ and its derivatives.

3.2.2 Partial Differentiation of a B-Spline Curve

To calculate the gradient of the objective function for optimization analytically, first we need to take partial derivatives of a B-spline curve with respect to the parameters, e.g. control points $c_\cdot$ and final time T. (If some of waypoint conditions q, $\dot{q}$, $\ddot{q}$ are free, they should be optimization parameters and we need to take partial derivatives of a B-spline with respect to them to calculate the gradient of objective function analytically. We skip these issues because it can be done by the similar procedure as partial derivatives with respect to control points.)

From (3.2.13), we split $\tilde{q}(\tilde{t})$ into two B-spline curves as follows:

$$\tilde{q}(\tilde{t}; \tilde{u}_{1:n}, c_{1:m}) = \tilde{q}_1(\tilde{t}; \tilde{u}_{1:n}, c_{1,1:m}) + \tilde{q}_2(\tilde{t}; \tilde{u}_{1:n}, c_{2,1:m}),$$

where $c_{1,\cdot}$ are computed by (3.2.12) with boundary conditions and final time T. (The reason why we split $\tilde{q}(\tilde{t})$ into two B-spline curves is to make explanation easier. With this expression, we can easily find the independence between optimization parameters and boundary conditions, which helps us implement the algorithm in terms of practical meaning.) Here, we can see that only

c_i $(i = 4, ...m-3)$ can be optimization parameters. Since we already know that $\tilde{u}_\cdot$ are defined by the number of knots and the order of a B-spline, the bases of the B-spline do not depend on the control points and final time T. Therefore $\tilde{q}_1$ just depend on final time T and $\tilde{q}_2$ depend on only control points c_i $(i = 4, ..., m-3)$. Let's take partial derivatives with respect to parameters:

$$\begin{aligned}
\frac{\partial \tilde{q}}{\partial T}(\tilde{t}) = \tilde{q}_{1,T}(\tilde{t}; \tilde{u}_{1:n}, \frac{\partial}{\partial T} c_{1,1:m}) \quad &= \sum_{i=1}^{m} \left(\frac{\partial}{\partial T} c_{1,i} \right) \tilde{B}_i(\tilde{t}) \\
\frac{\partial \tilde{q}}{\partial c_j}(\tilde{t}) = \tilde{q}_{2,c_j}(\tilde{t}; \tilde{u}_{1:n}, \frac{\partial}{\partial c_j} c_{2,1:m}) \quad &= \sum_{i=1}^{m} \left(\frac{\partial}{\partial c_j} c_{2,i} \right) \tilde{B}_i(\tilde{t}),
\end{aligned} \tag{3.2.19}$$

where

$$\frac{\partial}{\partial T} c_{1,i} = \begin{cases}
0 & \text{if } i = 1 \\
\frac{\Delta}{p-1} \dot{q}(0) & \text{if } i = 2 \\
\frac{3\Delta}{p-1} \dot{q}(0) + \frac{4T\Delta^2}{(p-1)(p-2)} \ddot{q}(0) & \text{if } i = 3 \\
0 & \text{if } i = 4, ..., m-3 \\
-\frac{3\Delta}{p-1} \dot{q}(T) + \frac{4T\Delta^2}{(p-1)(p-2)} \ddot{q}(T) & \text{if } i = m-2 \\
-\frac{\Delta}{p-1} \dot{q}(T) & \text{if } i = m-1 \\
0 & \text{if } i = m
\end{cases}$$

$$\frac{\partial}{\partial c_j} c_{2,i} = \begin{cases}
1 & \text{if } i = j = 4, ..., m-3 \\
0 & \text{otherwise.}
\end{cases}$$

Because control points and final time do not depend on normalized time variable $\tilde{t}$, it is possible that

$$\frac{\mathrm{d}\tilde{q}}{\mathrm{d}\tilde{t}} = \frac{\partial \tilde{q}}{\partial \tilde{t}}.$$

Therefore we can get partial derivatives of $\partial\tilde{q}/\partial\tilde{t}$ and $\partial^2\tilde{q}/\partial\tilde{t}^2$ with respect to the parameters, T and c_j, as follows:

$$\begin{aligned}
\frac{\partial}{\partial T}\frac{\partial\tilde{q}}{\partial\tilde{t}} &= \frac{\partial}{\partial\tilde{t}}\frac{\partial\tilde{q}}{\partial T} \\
\frac{\partial}{\partial c_j}\frac{\partial\tilde{q}}{\partial\tilde{t}} &= \frac{\partial}{\partial\tilde{t}}\frac{\partial\tilde{q}}{\partial c_j} \\
\frac{\partial}{\partial T}\frac{\partial^2\tilde{q}}{\partial\tilde{t}^2} &= \frac{\partial^2}{\partial\tilde{t}^2}\frac{\partial\tilde{q}}{\partial T} \\
\frac{\partial}{\partial c_j}\frac{\partial^2\tilde{q}}{\partial\tilde{t}^2} &= \frac{\partial^2}{\partial\tilde{t}^2}\frac{\partial\tilde{q}}{\partial c_j}
\end{aligned}, \quad j = 4, ..., m-3. \tag{3.2.20}$$

Finally, from (3.2.16), (3.2.17) and (3.2.19), (3.2.19), we can easily take partial derivatives of $q(t(\tilde{t}))$, $\dot{q}(t(\tilde{t}))$, and $\ddot{q}(t(\tilde{t}))$ with respect to the parameters.
As you can see above, we do not need to calculate partial derivatives with respect to control points every optimization steps. Taking partial derivatives with respect to control points only once before running optimization loop, we can reduce the repetition calculations.

3.3 Gaussian Quadrature

There are various kinds of numerical integration methods. Obviously, choosing a method is important because the performance of numerical integration method directly effects that of whole algorithm. Because calculating the function which we want to integrate takes some time, to reduce the computation time, integration method has to use the calculating points as few as possible without accuracy problem. In addition, because the domain of function we used is scalable, we need to handle it easily with integration method. These are the reasons why we use Gaussian quadrature for integration.
An n-point Gaussian quadrature states as a weighted sum of function values,

which is defined over $[-1, 1]$, at specified points:

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx \sum_{i=1}^{n} \omega_i f(x_i), \quad i = 1, ..., n.$$

There are several ways to get points x_i and weights w_i. We use the method called Gauss-Legendre quadrature which use Legendre polynomials [23]. Before using the Gaussian quadrature method in our problem (3.2.18), an integral over $[0, T]$ must be changed into an integral over $[-1, 1]$. Changing interval can be done by following way:

$$\begin{aligned}\int_0^T \tilde{E}(\tilde{t}(t))\mathrm{d}t &= T\int_0^1 \tilde{E}(\tilde{t})\mathrm{d}\tilde{t} \\ &= \frac{T}{2}\int_{-1}^{1} \tilde{E}\left(\frac{1}{2}\tilde{t} + \frac{1}{2}\right)\mathrm{d}\tilde{t}\end{aligned}$$

Applying the Gaussian quadrature method, we can get the following equation:

$$\int_0^T \tilde{E}(\tilde{t}(t))\mathrm{d}t = \frac{T}{2}\sum_{i=1}^{n} \omega_i \tilde{E}\left(\frac{1}{2}\tilde{t}_i' + \frac{1}{2}\right),$$

where $(\omega_i, \tilde{t}_i')$ are the weights and the points of an n-point Gaussian quadrature. Selecting n depends on the purpose of the program. For our case, we want to get the result of optimization in $500ms$ in our environment. Heuristically we find that $25 \sim 35$ is suitable for n.

3.4 Algorithm for Generating Energy-Efficient Trajectory

Parameterizing the joint trajectories of each sector and applying the previous results the optimization problem is transformed in the followings:

$$\begin{aligned}
\underset{c_{4:m-3},q,\dot{q},\ddot{q},T}{\text{Minimize}} \quad & J = \sum_{i=1}^{W-1} \frac{T_i}{2} \sum_{j=1}^{n} \omega_j \tilde{E}_i \left(\frac{1}{2}\tilde{t}'_j + \frac{1}{2} \right) \\
\text{subject to} \quad & 0 \le T_i \\
& q_{min} \le c_{i,k} \le q_{max} \\
& \dot{q}_{min} \le \frac{1}{T} c'_{i,k}(c_i) \le \dot{q}_{max} \\
& \ddot{q}_{min} \le \frac{1}{T^2} c''_{i,k}(c_i) \le \ddot{q}_{max} \\
& \tau_{min} \le \tilde{\tau}_i \left(\frac{1}{2}\tilde{t}'_j + \frac{1}{2} \right) \le \tau_{max},
\end{aligned} \tag{3.4.21}$$

where

$$\begin{aligned}
T_i &= t_{i+1} - t_i \quad : \text{traveling time of sector } i \\
\tau(q,\dot{q},\ddot{q}) &= M(q)\ddot{q} + C(q,\dot{q})\dot{q} + N(q,\dot{q}) \\
\tilde{q}_i(\tilde{t}) &= \tilde{q}_i(\tilde{t};\tilde{u}_i,c_i) = \sum_{k=1}^{m} c_{i,k}\tilde{B}_k(\tilde{t}), \quad \tilde{t} \in [0,1] : \text{B-spline} \\
c_{i,k},\ c'_{i,k},\ c''_{i,k} &= k^{\text{th}} \text{ control point of } \tilde{q}_i,\ \frac{\mathrm{d}\tilde{q}_i}{\mathrm{d}\tilde{t}},\ \frac{\mathrm{d}^2\tilde{q}_i}{\mathrm{d}\tilde{t}^2}, \text{ respectively} \\
\tilde{\tau}_i(\tilde{t}) &= \tau\left(\tilde{q}_i(\tilde{t}), \frac{1}{T_i}\frac{\mathrm{d}\tilde{q}_i}{\mathrm{d}\tilde{t}}(\tilde{t}), \frac{1}{T_i^2}\frac{\mathrm{d}^2\tilde{q}_i}{\mathrm{d}\tilde{t}^2}(\tilde{t}) \right) \\
\tilde{E}_i(\tilde{t}) &= E(\tilde{\tau}_i(\tilde{t})) \\
(w_j, \tilde{t}'_j) &= j^{\text{th}} \text{ weight and point of } n\text{-points Gaussian Quadrature.}
\end{aligned}$$

As we already discussed, some of control points are computed by boundary conditions (3.2.12). Note that waypoint conditions which are free, t_i, q_i, $\dot{q}_i$, and $\ddot{q}_i$, are included as optimization parameters. By convex hull property of

B-spline (3.2.9), joint position, velocity, and acceleration limits can be formed as linear inequality conditions in parameter space (3.2.8) [10]. In contrast, because joint torque limits are highly nonlinear, we cannot do optimization with hard constraints on joint torque at every time. So, we check torque limit on specific times.

The gradients of the objective function is:

$$\nabla J = \begin{bmatrix} \frac{\partial J}{\partial c_{i,k}} & \frac{\partial J}{\partial q_i} & \frac{\partial J}{\partial \dot{q}_i} & \frac{\partial J}{\partial \ddot{q}_i} & \frac{\partial J}{\partial T_i} \end{bmatrix}$$

where can be computed from the analytical results in previous sections. Again, if some waypoint conditions are given, we do not need to take into account in optimization parameters. Calculating numerical gradient of the objective function using first symmetric derivative,

$$\lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h},$$

takes about 13 times as much time as calculating its analytical gradient.

During every optimization step, we compute function values and gradients of the objective function and inequality conditions. In this process, some calculations are duplicated such as calculating partial derivatives of B-splines and joint torques with respect to optimization parameters. Reusing them when it occurs, we find that the computation time decreases dramatically.

3.5 Case Studies

In this section, we demonstrate optimal trajectory generations for an Efort robot whose dynamic properties are given. The optimal trajectory generations we performed are roughly divided into three cases. For the first case, we use the effort function as an energy consuming function $E(\tau)$. Using the effort function, we generate the trajectory from given point to another. We

measure the computation time and compare the objective function value of our algorithm to that of Reflexxes which is a S-curve based time-optimal trajectory generation. Also, we generate the trajectory with waypoints and payloads. Second, we use the energy loss function. Similarly, we compare our algorithm's results to Reflexxes' and generate the trajectory with waypoints and payload. In the last case, we generate the energy-optimal trajectory with base link optimization, which means that we want to find the optimal place to locate the robot while minimizing the objective function. In the factory, replacing the industrial robot is uncommon and the task may be fixed. Finding a good place to locate the robot is important with fixed traveling time. We augment the optimization parameters with six generalized coordinates for representing the position and the orientation of the base link. We will explain this case in more details in the corresponding section. The optimization problem formulated in Section 3.4 is solved by optimization tool called NLopt [24] which is a free open source library for nonlinear optimization. All of the optimizations were performed on Intel(R) Core(TM) i7-6700 CPU processor @ 3.40GHz and 16.0GB RAM with 64bit Windows 10 Education K.

Table 3.1: Kinematics and dynamics constraints for each joint of an Efort robot

Joint no. i	Pos. q_i (rad)		Vel. $\dot{q}_i$ (rad/s)		Acc. $\ddot{q}_i$ (rad/s^2)		Torque τ_i (Nm)	
	min	max	min	max	min	max	min	max
1	-3.054	3.054	-1.745	1.745	-8.726	8.726	-3000	3000
2	-1.745	1.570	-1.396	1.396	-6.981	6.981	-3000	3000
3	-2.530	1.221	-2.443	2.443	-12.21	12.21	-1530	1530
4	-3.141	3.141	-5.061	5.061	-25.30	25.30	-369.3	369.3
5	-2.356	2.356	-5.061	5.061	-25.30	25.30	-384	384
6	-6.283	6.283	-7.679	7.679	-38.39	38.39	-143.2	143.2

Before we talk about three cases, first we will show you how we select the hyper-parameters for B-spline such as the order of B-spline and the number of control points. Because there are joint acceleration boundary conditions, joint trajectories must be the second order differentiable or more. For that reason, we use the fourth order of B-spline curve. Now, let's talk about the number of control points. To choose proper number of control points, we investigate objective function value, which is defined as (3.1.2) with the effort function, and computation time versus the number of control points as in Figure 3.3. As you can see Figure 3.3, the objective function value tends to decrease as

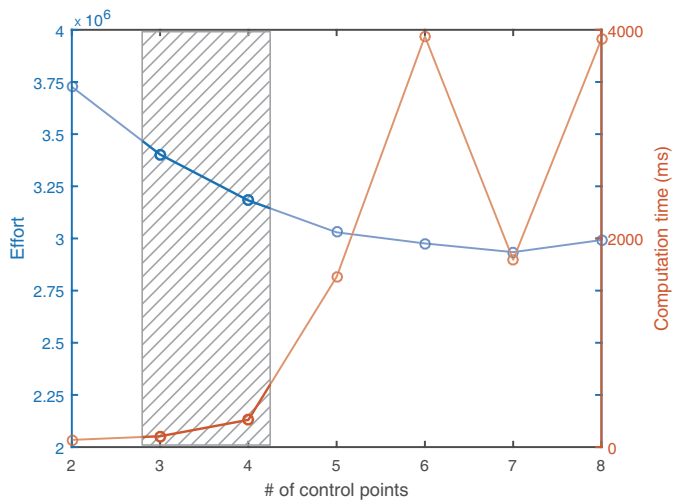


Figure 3.3: Objective function values and computation times versus the number of control points. The optimization is performed without waypoint and payload.

the number of control points increases. In contrast, the computation time tends to increase as the number of control points increases. Because selecting hyper-parameters depend on the purpose of the program, we think three or four control points for each B-spline is suitable for us. In our simulations, we

use four control points. In addition, we use a 30-point Gaussian quadrature and all of the optimizations are performed with the first three joints, not six joints, because when we use six joints for the optimization, the optimization takes more time and the result is even worse than when we use first three joints.

3.5.1 Effort

These simulations are performed with the library we made in C++ language. The initial, final, and waypoint configurations are given as in Figure 3.4. The optimization is done with the effort function. Effort case is largely divided into two small cases.

First, we performed the optimization with only the initial and the final configurations. It means that there is no waypoint and payload. The optimal trajectories are shown in Figure 3.5. The black lines represent the trajectories of the end effector of the robot. Figure 3.5(a) shows the optimal trajectory with given final time and Figure 3.5(b) shows the results of the optimization with free final time. The numerical results are expressed in Table 3.2. As you can see, we can get the optimal trajectory within short time. When the final time is free, it takes more time but the function value is less than when the final time is fixed. Note that there is no optimization in Reflexxes, so it takes less time to generate the trajectory. The objective function value from our optimal trajectory is much less than the objective function value of Reflexxes' trajectory. Generated position, velocity, acceleration, and torque of each joint are in Figure 3.6.

Second, we performed the optimization with the waypoints and payloads on the end effector. At waypoint, the joint velocity and acceleration are free. The time when the robot is passing through the waypoint is also free. Inertia of

the payload is defined as follows:

$$I = 70 * I^{6\times6},$$

where $I^{6\times6}$ is a 6 by 6 identity matrix. The motions are shown in Figure 3.7 and the numerical results are expressed in Table 3.3. we present generated position, velocity, acceleration, and torque of each joint in Figure 3.8. The dash line shows the time when the robot pass through the given waypoint.

3.5.2 Energy Loss

These simulations are performed with the energy loss function. The initial, final, and waypoint configurations are given as in Figure 3.4. Like effort case, this case is also divided into two small cases. For the case without waypoint and payload, the simulation results are expressed in Table 3.9 and Figure 3.4. Generated position, velocity, acceleration, and torque for each joint are represented in Figure 3.10. For the case with waypoint and payload, the simulation results are expressed in Table 3.11 and Figure 3.5. Again, at the waypoint, the joint velocity, the acceleration and the waypoint time are free. Generated position, velocity, acceleration, and torque for each joint are represented in Figure 3.12. In contrast with effort case, potential energy difference between two configurations, initial and final, is the dominant factor of the objective function value. This is the reason why the difference between the results of our algorithm and Reflexxes is small.

3.5.3 Base Link Optimization

In terms of practical meanings, time-optimal trajectory is more important than energy-optimal trajectory in many cases. Most of users want to reduce the energy consumption without increasing traveling time. Optimizing the

location of the base link could be one of the solutions, which means that we want to find the location of the base link to minimize the energy consumption of the robot for certain motion without increasing traveling time. The reason why it is more meaningful and more realizable is that the motion and the location of the robot are usually fixed for industrial robots. Here, we assume that the joint velocity and the acceleration at initial and final points are fixed as zero. (The joint velocity and acceleration at other points are free.) To solve this problem, we express the position and the orientation of the base link as six generalized coordinates and then optimize the objective function. For example, the position and the orientation of the base link can be expressed as follows:

$$T_{base}(\theta) = \prod_{i=1}^{6} e^{[s_i]\theta_i},$$

where $\theta \in \mathbb{R}^6$ is a six-generalized coordinates and $T_{base}(\theta)$ is $SE(3)$ of the base link, s_i is a ith screw and $[s_i]$ is corresponding $se(3)$. Therefore, we can get the end effector frame T_{ee} as follows:

$$T_{ee}(\theta, q) = T(\theta)T_{robot}(q),$$

where $T_{robot}(q)$ is a function of q (joint values) which represents forward kinematics of the robot. Because the position and the orientation at given waypoint are fixed, we can easily get

$$J(\theta, q) \begin{bmatrix} \mathrm{d}\theta \\ \mathrm{d}q \end{bmatrix} = 0,$$

where J is a body Jacobian which can be defined as

$$J(\theta, q) = \begin{bmatrix} J_\theta & | & J_q \end{bmatrix} = \begin{bmatrix} T_{ee}^{-1}\frac{\partial T_{ee}}{\partial \theta} & | & T_{ee}^{-1}\frac{\partial T_{ee}}{\partial q} \end{bmatrix}.$$

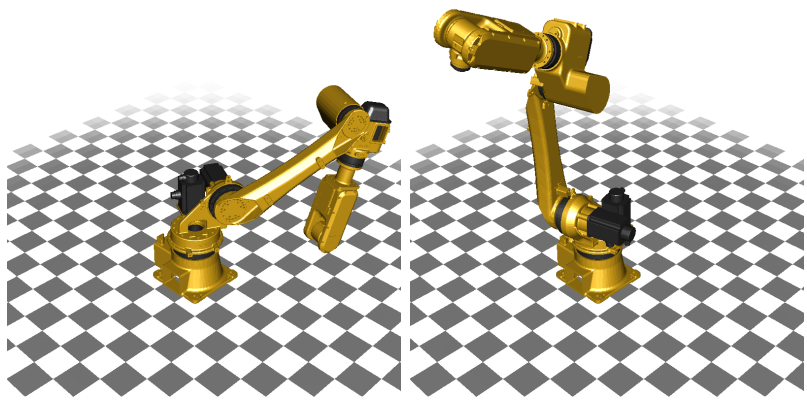
We expand the above equation and then we get

$$0 = J_\theta \mathrm{d}\theta + J_q \mathrm{d}q$$
$$\frac{\mathrm{d}q}{\mathrm{d}\theta} = -J_q^{-1} J_\theta.$$

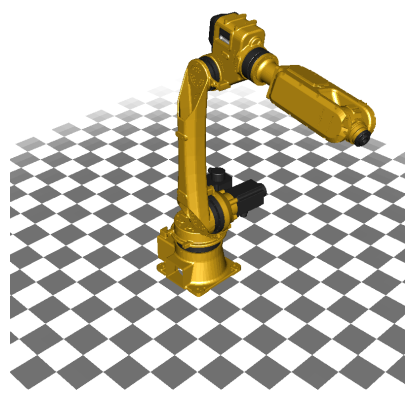
Note that, because the joint velocity and the acceleration at initial and final points are fixed as zero, $\mathrm{d}\dot{q}/\mathrm{d}\theta$ and $\mathrm{d}\ddot{q}/\mathrm{d}\theta$ are also equal to zero. In addition, the orientation of the base link affects the spatial acceleration of base link which is used in inverse dynamics. Using above results, we can calculate the partial derivative of the objective function with respect to θ analytically:

$$\frac{\partial \tau}{\partial \theta} = \frac{\partial \tau}{\partial \dot{V}} \frac{\partial \dot{V}}{\partial \theta} + \frac{\partial \tau}{\partial q} \frac{\partial q}{\partial \theta} + \frac{\partial \tau}{\partial \dot{q}} \frac{\partial \dot{q}}{\partial \theta} + \frac{\partial \tau}{\partial \ddot{q}} \frac{\partial \ddot{q}}{\partial \theta}.$$

Simulation results are expressed in Table 3.6 and Figure 3.13. As you can see, even small displacement of the base link makes about 3%∼10% reduction in energy consumption in terms of both effort and energy loss.

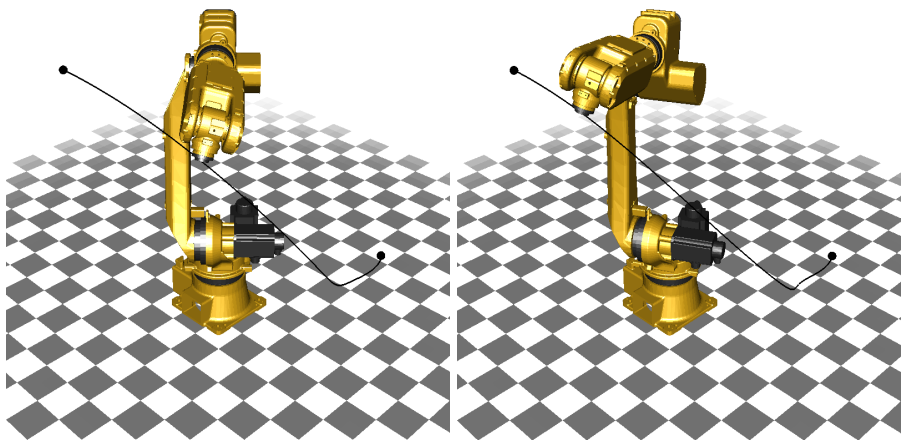


(a) Initial configuration (b) Final configuration



(c) Configuration at given waypoint

Figure 3.4: Configurations at start point, end point and waypoint



(a) Optimal trajectoy with fixed final time

(b) Optimal trajectoy with free final time

Figure 3.5: Optimal trajectory for the first case (effort function) without waypoint and payload

Table 3.2: Optimal results for the first case (effort function) without waypoint and payload

waypoint(x) payload($0kg$)		Final time t_f (s)	Computation Time (ms)	Obj. Function Value
ours	t_f given	2.00	66	692777
	t_f free	1.85	166	689421
Reflexxes'		1.58	1	1104480

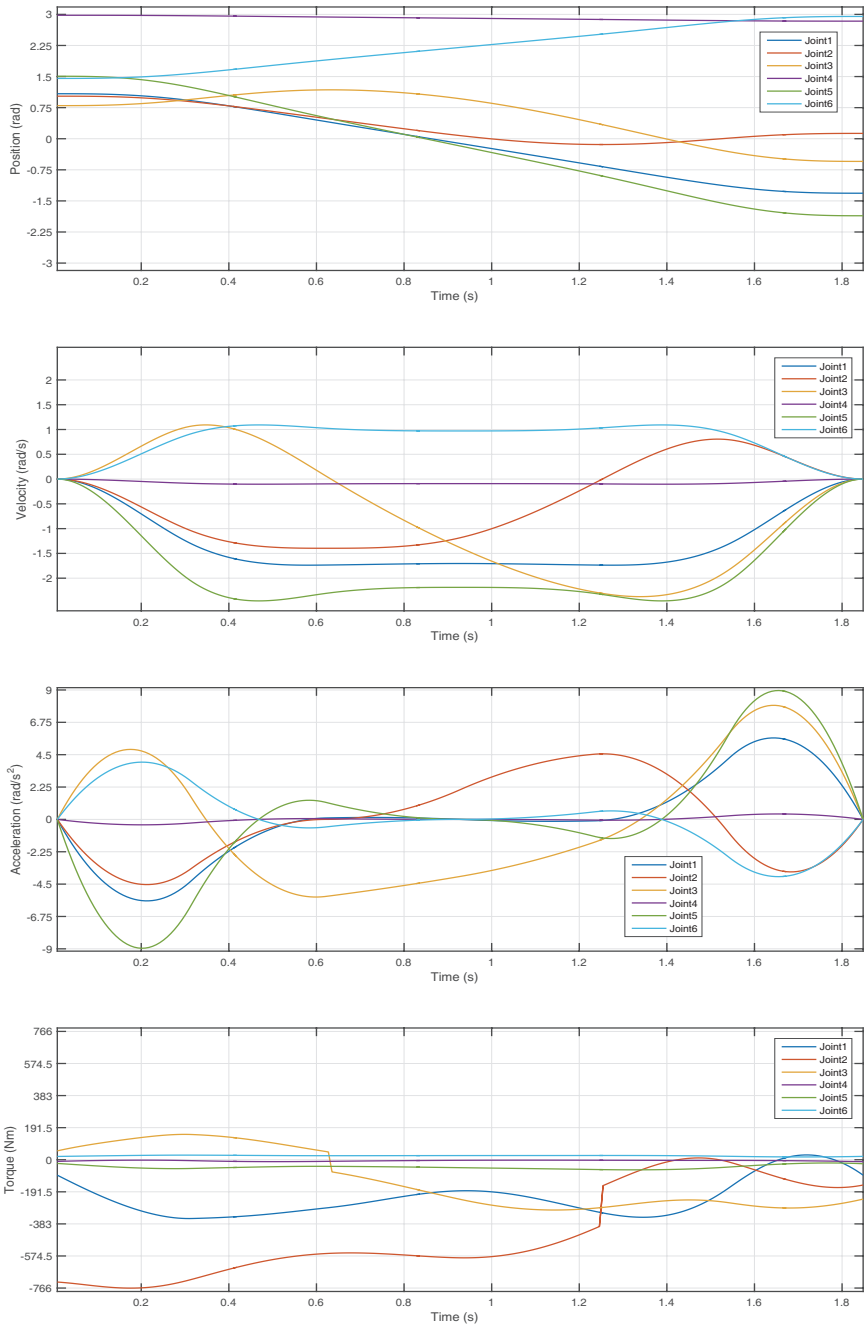
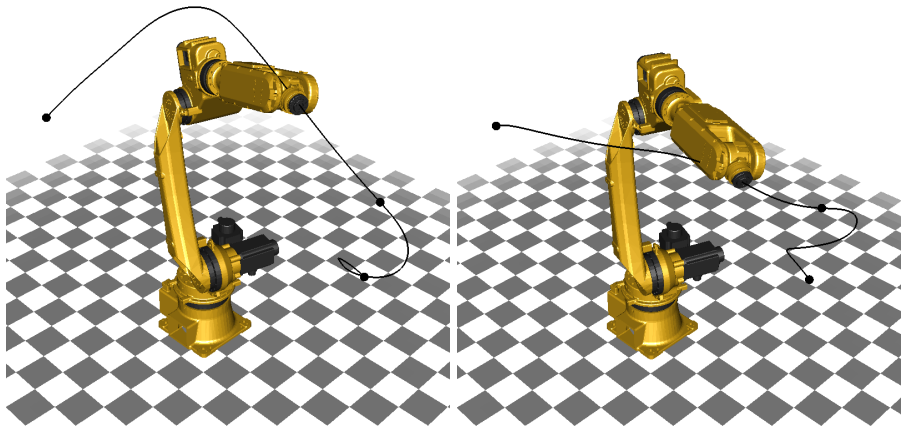


Figure 3.6: Position, velocity, acceleration, and torque of each joint for the first case (effort function) with free final time (no waypoint, no payload)



(a) Optimal trajectoy with fixed final time

(b) Optimal trajectoy with free final time

Figure 3.7: Optimal trajectory for the first case (effort function) with waypoints and payloads

Table 3.3: Optimal results for the first case (effort function) with waypoints and payloads

waypoint(o) payload($70kg$)		Final time t_f (s)	Computation Time (ms)	Obj. Function Value
ours	t_f given	5.00	1518	5180882
	t_f free	3.55	5891	4524667

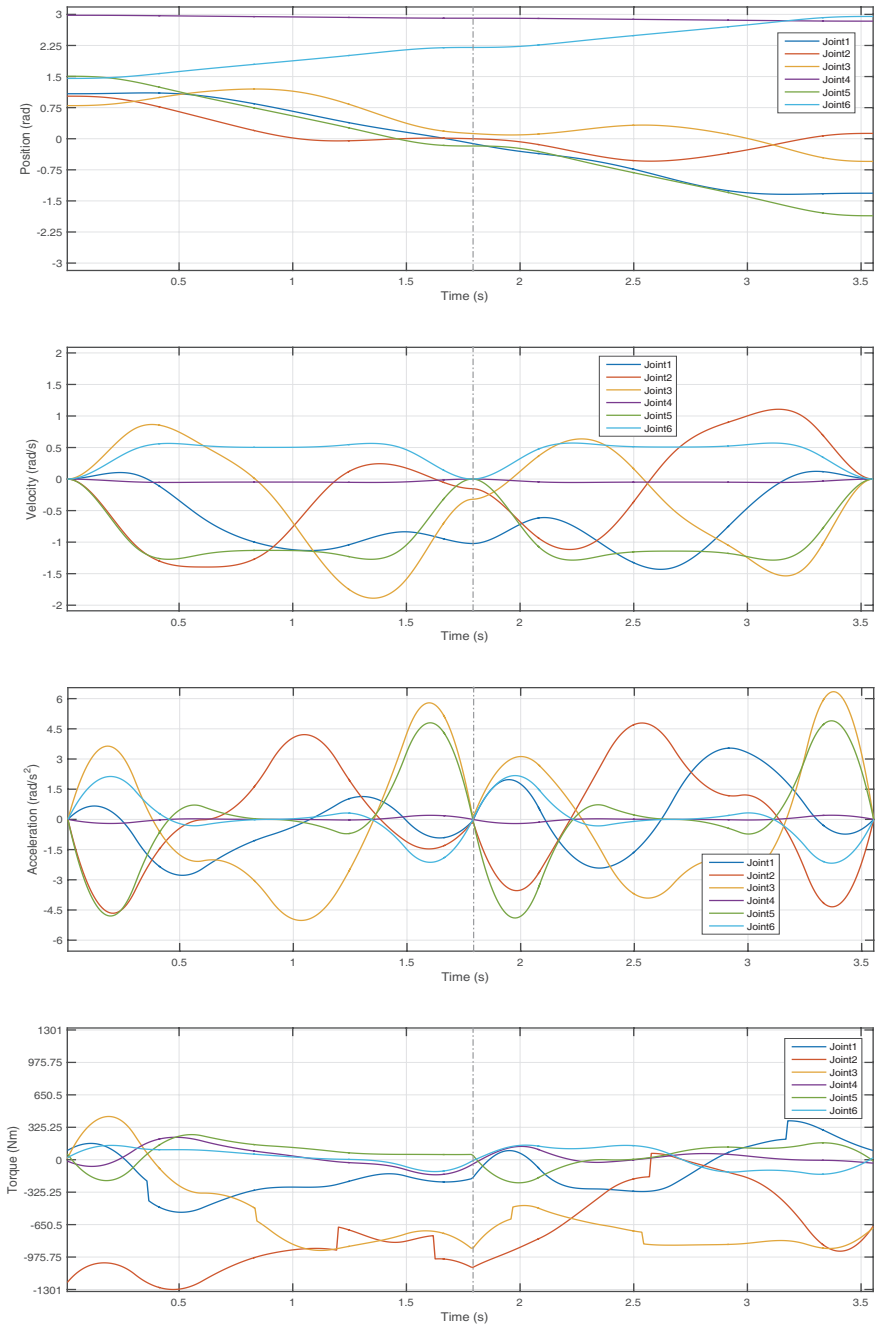
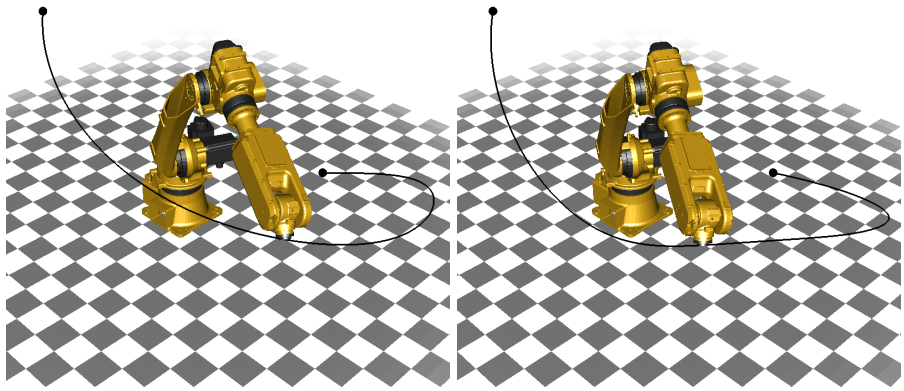


Figure 3.8: Position, velocity, acceleration, and torque of each joint for the first case (effort function) with free final time (include waypoints and payloads)



(a) Optimal trajectoy with fixed final time (b) Optimal trajectoy with free final time

Figure 3.9: Optimal trajectory for the second case (energy loss function) without waypoint and payload

Table 3.4: Optimal results for the second case (energy loss function) without waypoint and payload

waypoint(x) payload($0kg$)		Final time t_f (s)	Computation Time (ms)	Obj. Function Value
ours	t_f given	2.00	197	1863.17
	t_f free	4.14	210	1671.86
Reflexxes'		1.58	1	1947.72

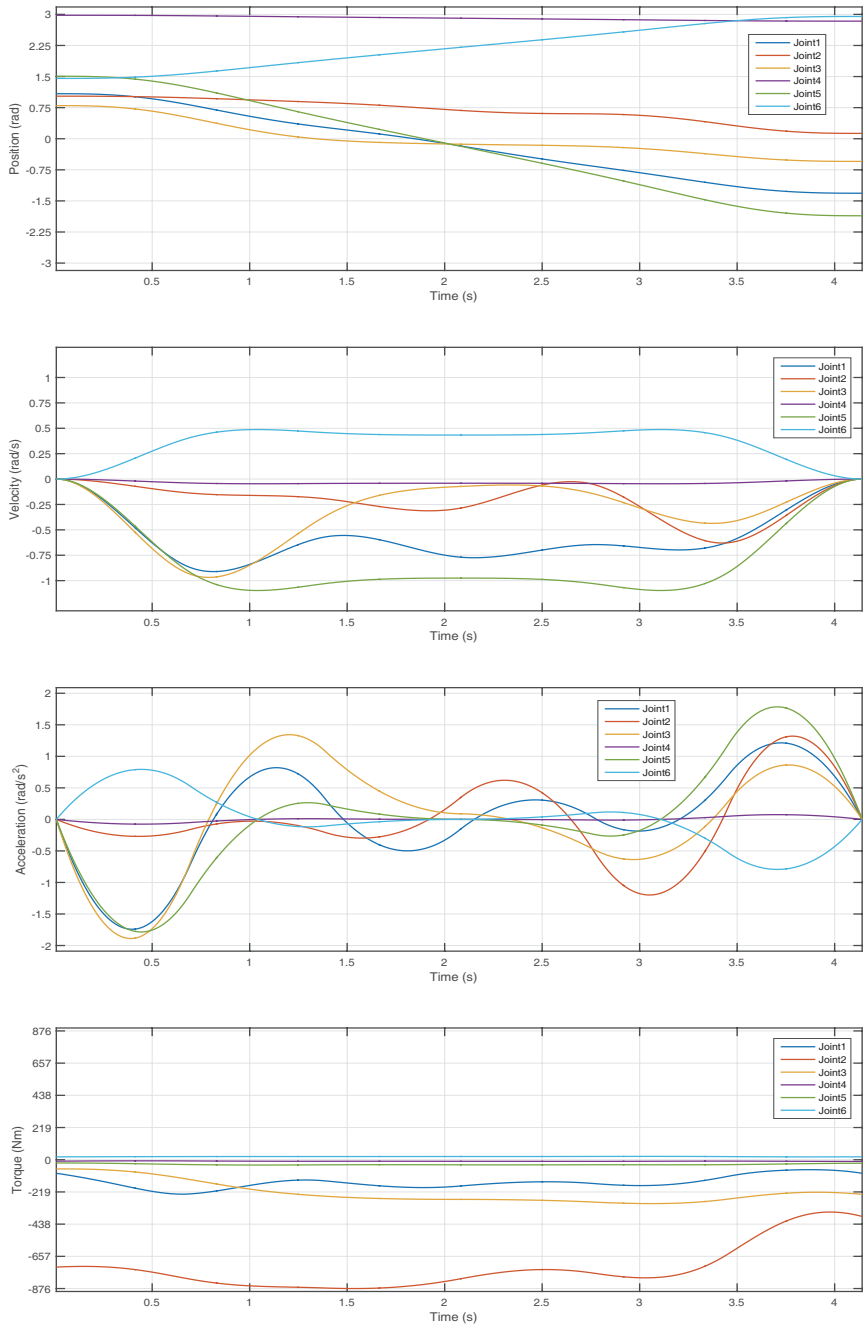
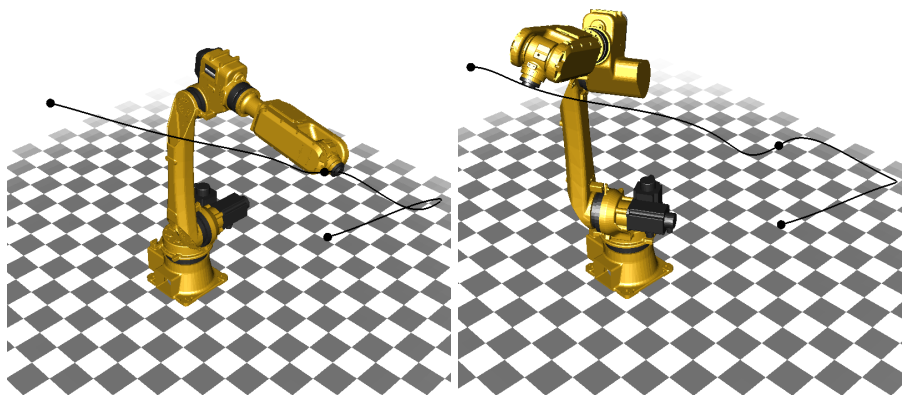


Figure 3.10: Position, velocity, acceleration, and torque of each joint for the second case (energy loss function) with free final time (no waypoint, no payload)



(a) Optimal trajectoy with fixed final time

(b) Optimal trajectoy with free final time

Figure 3.11: Optimal trajectory for the second case (energy loss function) with waypoints and payloads

Table 3.5: Optimal results for the second case (energy loss function) with waypoints and payloads

waypoint(o) payload($70kg$)		Final time t_f (s)	Computation Time (ms)	Obj. Function Value
ours	t_f given	5.00	880	3719.55
	t_f free	4.77	2170	3615.58

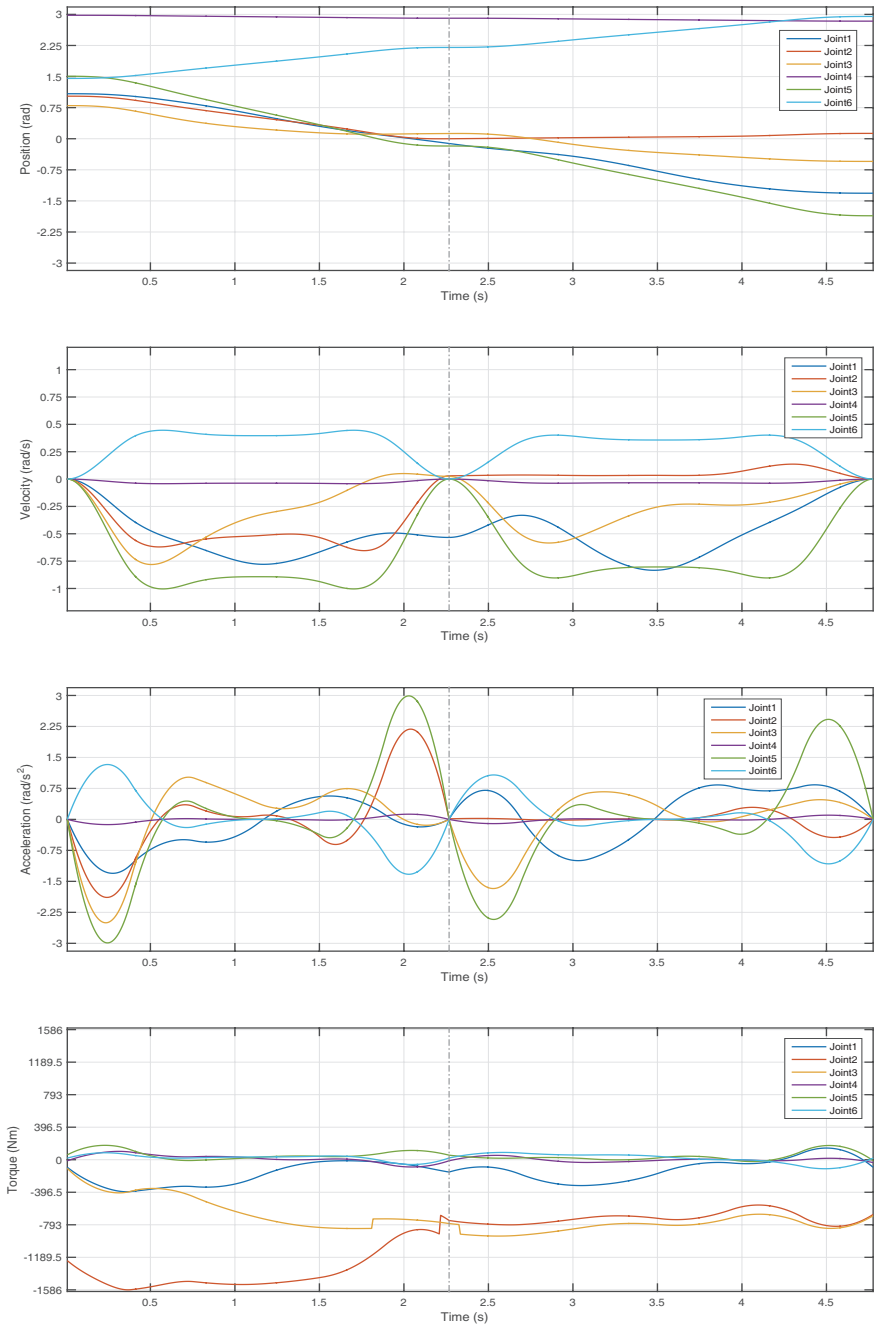
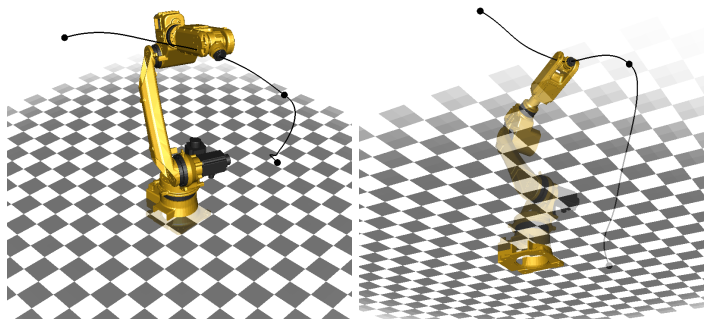


Figure 3.12: Position, velocity, acceleration, and torque of each joint for the second case (energy loss function) with free final time (include waypoints and payloads)

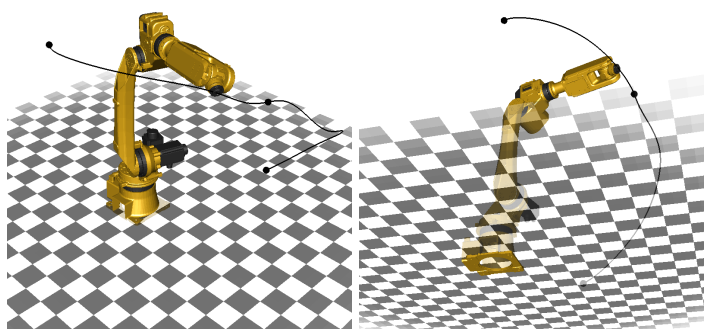
Table 3.6: Results of base link optimization

Objective Function Type	Without base link optimization	With base link optimization	Remarks
Effort (t_f given)	5180882	4639428	10.45% ↓
Energy loss (t_f given)	3719.55	3585.98	3.59% ↓



(a) Optimization motion from top view (the effort function)

(b) Optimization motion from bottom view (the effort function)



(c) Optimization motion from top view (the energy loss function)

(d) Optimization motion from bottom view (the energy loss function)

Figure 3.13: Optimal position and orientation of the base link using two objective functions

4

Task Scheduling of Energy-Optimal Trajectories

In this section, a task scheduling algorithm is presented which minimizes the energy consumption. As we said in Section 1, many operators are interested in the time optimality, but sometimes the time-optimal operation cannot be used or it is less important, for example, an assembly line with bottleneck of a pass-or-fail task such as pick-and-place task. In pick-and-place task, the robot should have to pick all objects and place them within certain corresponding time range for each object. In other words, for such like problems, reducing complete time is not an objective. The robots do not need to move as fast as they can. Using residual time, we can minimize the energy consumption. The algorithm which will be described in this section can be the one of solutions to solve these kinds of problems with some assumptions which are acceptable in terms of practical meanings. We use the dynamic programming method with the energy-efficient trajectory generation algorithm which we described in previous section.

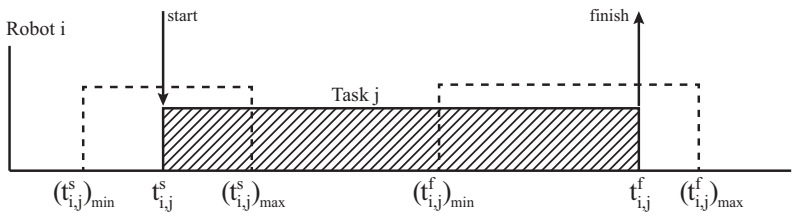


Figure 4.1: Task timeline for explaining some parameters

4.1 Problem Definition

Let's n and m are the numbers of robots and tasks, respectively. Each pair of a robot and a task has information $\mathcal{I}_{i,j} = \left(C_{i,j}, (t^s_{i,j})_{\min,\max}, (t^f_{i,j})_{\min,\max}, d_{i,j}\right)$, see Figure 4.1: ($i$ is the index of a robot, j is the index of a task)

$$
\begin{aligned}
C_{i,j}(t^s, t^f): &\text{ the cost when the robot } i \text{ performs the task } j \text{ from } t^s \text{ to } t^f \\
(t^s_{i,j})_{\min}, (t^s_{i,j})_{\max}: &\text{ the robot } i \text{ have to start the task } j \text{ from } (t^s_{i,j})_{\min} \text{ to } (t^s_{i,j})_{\max} \\
(t^f_{i,j})_{\min}, (t^f_{i,j})_{\max}: &\text{ the robot } i \text{ have to finish the task } j \text{ from } (t^f_{i,j})_{\min} \text{ to } (t^f_{i,j})_{\max} \\
d_{i,j}: &\text{ the minimum execution time of the task } j \text{ in robot } i,
\end{aligned}
$$

where $C_{i,j} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, $(t^s_{i,j})_{\min,\max}, (t^f_{i,j})_{\min,\max} \in \mathbb{R}$. If the robot i cannot perform the task j, there is no corresponding information $\mathcal{I}_{i,j}$. As you can infer from the above information, for our problem there is no precedence relationship between tasks, which means that all tasks are independent. To define the problem, we introduce some parameters as follows:

$$
\begin{aligned}
\omega_j: &\text{ the index of a robot where the task } j \text{ is performed } (\omega_j \in \mathbb{N}, 1 \leq \omega_j \leq n) \\
\mathcal{T}_i: &\text{ a set of tasks which are performed in robot } i \, (\mathcal{T}_i = \{j | \omega_j = i\}) \, .
\end{aligned}
$$

The proposed mathematical model for minimizing of the energy consumption is as follows:

$$\begin{aligned} \underset{\omega_j, t^s_{\omega_j,j}, t^f_{\omega_j,j}}{\text{Minimize}} \quad & J = \sum_{j=1}^{m} C_{\omega_j,j}(t^s_{\omega_j,j}, t^f_{\omega_j,j}) \\ \text{subject to} \quad & (t^s_{\omega_j,j})_{\min} \leq t^s_{\omega_j,j} \leq (t^s_{\omega_j,j})_{\max} \\ & (t^f_{\omega_j,j})_{\min} \leq t^f_{\omega_j,j} \leq (t^f_{\omega_j,j})_{\max} \\ & t^f_{\omega_j,j} - t^s_{\omega_j,j} \geq d_{\omega_j,j}, \end{aligned} \tag{4.1.1}$$

where cost function $C_{\omega_j,j}$ represents the consumed energy which the robot w_j use when it performs the task j. In addition to above constraints, because it is impossible that a robot performs multiple tasks in a robot at the same time, we need to consider the corresponding constraints [12] (mutual exclusion). This issue can be expressed in the following way. Let's define σ_{j_1,j_2} as a Boolean variable representing task j_1 being performed after task j_2. If two tasks, j_1 and j_2, are performed in the same robot ($i = \omega_{j_1} = \omega_{j_2}$), then only one of σ_{j_1,j_2} and σ_{j_2,j_1} will be one (true) and, of course, the other is zero (false). The resulting constraints can be expressed as

$$\begin{aligned} & t^s_{\omega_{j_1},j_1} \geq t^f_{\omega_{j_2},j_2} - M(1 - \sigma_{j_1,j_2}) - M|\omega_{j_1} - \omega_{j_2}| \\ & t^s_{\omega_{j_2},j_2} \geq t^f_{\omega_{j_1},j_1} - M(1 - \sigma_{j_2,j_1}) - M|\omega_{j_1} - \omega_{j_2}| \\ & \sigma_{j_1,j_2} + \sigma_{j_2,j_1} = 1, \end{aligned} \tag{4.1.2}$$

where M is sufficiently large for negating constraints if they are unnecessary. As we mentioned above, these constraints are active when the robots for task j_1 and j_2 are same. And also if task j_1 is performed after task j_2, then σ_{j_1,j_2} is one (true) and the first constraint is valid. Of course, the variable σ will be optimized. If there are $\omega_j, t^s_{\omega_j,j}, t^f_{\omega_j,j}$ for all j, and σ which satisfy above constraints (4.1.1), (4.1.2), we can say that the tasks are *Schedulable* with the given robots.

4.2 Assumptions

Now, let's think about how we can solve this kind of problems. Note that the optimization variables are the indices of the robot for each task which are natural numbers ω, the real valued starting and finishing times for each task t^s, t^f, and the binary variables representing mutual exclusion σ. All things considered, we can solve this problem using mixed integer programming. This problem is, however, too complex to be solved and it takes so many times to solve it. If it is solved, the result may be local minimum and not really good. In other words, solving this problem directly is not useful. We think that the problem definition is too general to get more meaningful result, so several acceptable assumptions are needed to make the problem more practical. In this section, we describe four assumptions. It may be hard to apply some of assumptions directly for some situations or factories. However, we think that, even if we force these assumptions to be applied in such situations and factories, the result won't be changed much in many cases.

The first assumption is about the final time constraints, $t^f_{\min}, t^f_{\max}$. In many cases, there is no final time limitation. If it exists, final time limitations may be for co-working with the machines which are not in the system we are dealing with. For example, there is a press machine which presses a material frequently. Here, we need to make an optimal schedule and trajectories of robots to pick the materials and place them into the press machines in time. In other words, for interaction between the robots and the press machine, we have to apply the final time limitations for the robots in the optimization. In contrast, if all robots and machines are under control and optimized together, final time limitations may not be needed in many cases.

Assumption 4.2.1. Final time limitations for each task do not exist.

$$(t^f_{i,j})_{\min} \to -\infty, (t^f_{i,j})_{\max} \to \infty.$$

In our problem, although there is no explicit final time limitation, the algorithm solves our problem like with final time constraints. We remark that the purpose of our algorithm is to minimize the energy consumption of robots while every tasks are performed. Because each task has the available time range for being started, the robot is forced to finish the previous task to start the task in time. However, the task which will be executed at the end has no limitation for finishing. To avoid this, we apply a final complete time, which means that all tasks are finished in certain time. (This constraint will be discussed in the next section.)

Assumption 4.2.2. Each robot can execute only one type of task.

$$\begin{aligned} C_i(t^s, t^f) &= C_{i,j}(t^s, t^f) \\ \Delta_i &= (t^s_{i,j})_{\max} - (t^s_{i,j})_{\min}, \qquad \text{for all } i,j \text{ such that } \mathcal{I}_{i,j} \text{ exists}, \\ d_i &= d_{i,j} \end{aligned}$$

where $C_i : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and $\Delta_i, d_i \in \mathbb{R}$.

Second, we assume that a robot can perform only one type of task. The tasks in the same type have same properties. Here, the properties are the cost function, the start time range, and minimum execute time of the task. Let's imagine a robot with a conveyor belt. The task is to pick and place the object moving on the conveyor belt. Because the area of activity of the robot is limited and the speed of the conveyor belt is constant, the available time interval for picking the object is fixed. In addition, because the inertia of the object is similar, we can think the minimum execution time for moving the object is also similar.

Assumption 4.2.3. Cost function depends on only the task execution time.

$$\mathcal{C}_{i,j}(t^f - t^s) = C_{i,j}(t^s, t^f), \quad \text{for all } i, j \text{ such that } \mathcal{I}_{i,j} \text{ exists,}$$

where $\mathcal{C}_{i,j} : \mathbb{R} \to \mathbb{R}$.

Next, we assume that for each pair of a robot and task the cost function depends on only the task execution time. We can apply this assumption when the task does not depend on the start time and the final time. For example, if the task is a point-to-point motion, the start point and the final point are fixed, so we can say that this situation can be solved with the above assumption. From Assumption 4.2.2 and Assumption 4.2.3, we can easily find that $\mathcal{C}_i : \mathbb{R} \to \mathbb{R}$ for $i = 1, ...n$ exist such that

$$\mathcal{C}_i(t^f - t^s) = C_{i,j}(t^s, t^f), \quad \text{for all } i, j \text{ such that } \mathcal{I}_{i,j} \text{ exists.}$$

Before we talk about the last assumption, let's see two important propositions which will be useful properties for our algorithm and can be proven with these three assumptions.

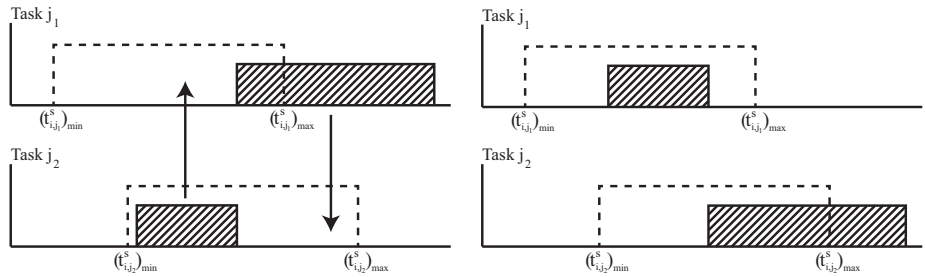
Proposition 4.1 (Schedulability)**.** *If the tasks are Schedulable with some robots, the tasks can also be Schedulable with the following constraint:*

$$t^f_{i,j_1} \leq t^s_{i,j_2}, \quad \textit{if } (t^s_{i,j_1})_{\min} \leq (t^s_{i,j_2})_{\min} \quad (\forall j_1, j_2 \in \mathcal{T}_i). \tag{4.2.3}$$

Proof. It can be proven by contradiction. Let the tasks are *Schedulable* while they do not satisfy the above constraint. There must be one or more pairs $(j_1, j_2), (j_1, j_2 \in \mathcal{T}_i)$ such that

$$t^f_{i,j_1} > t^s_{i,j_2}, \quad (t^s_{i,j_1})_{\min} \leq (t^s_{i,j_2})_{\min} \tag{4.2.4}$$

which means that the task j_2 is performed before the task j_1 is started. From



(a) Original schedule

(b) New schedule by switching the order of task j_1 and task j_2

Figure 4.2: Original schedule and rescheduling

the constraint (4.1.1), we can get

$$\begin{aligned} & t^f_{i,j_2} \le t^s_{i,j_1} \\ \Rightarrow & (t^s_{i,j_1})_{\min} \le (t^s_{i,j_2})_{\min} \le t^s_{i,j_2} \le t^f_{i,j_2} \le t^s_{i,j_1} \le (t^s_{i,j_1})_{\max} \end{aligned}$$

and, from Assumption 4.2.2, we can easily find

$$(t^s_{i,j_1})_{\max} = (t^s_{i,j_1})_{\min} + \Delta_i \le (t^s_{i,j_2})_{\max} = (t^s_{i,j_2})_{\min} + \Delta_i,$$

then,

$$\begin{aligned} & (t^s_{i,j_2})_{\min} \le t^s_{i,j_1} \le (t^s_{i,j_2})_{\max} \\ & (t^s_{i,j_1})_{\min} \le t^s_{i,j_2} \le (t^s_{i,j_1})_{\max}. \end{aligned}$$

Switching the order of the task j_1 and the task j_2, we can make another schedule, see Figure 4.2:

$$\begin{aligned} & \sigma_{j_1,j_2} = 1,\ \sigma_{j_2,j_1} = 0 \\ & (t^s_{i,j_1})_{\text{new}} = t^s_{i,j_2}, \qquad (t^f_{i,j_1})_{\text{new}} = t^f_{i,j_2} \\ & (t^s_{i,j_2})_{\text{new}} = t^s_{i,j_1}, \qquad (t^f_{i,j_2})_{\text{new}} = t^f_{i,j_1}. \end{aligned}$$

Using the same procedure, we can sort the tasks such that there is no pair $(j_1, j_2), (j_1, j_2 \in \mathcal{T}_i)$ which satisfies (4.2.4). Therefore, Proposition 4.1 must be true. □

and

Proposition 4.2 (Optimality)**.** *Let the tasks be Schedulable. If there are optimal solutions, at least one of them satisfies the constraint* (4.2.3)*.*

Proof. Suppose that there is no optimal solution subject to the constraint (4.2.3). Let's pick another optimal solution and then using the method we described at the proof of Proposition 4.1 we can sort the task to satisfy the constraint (4.2.3). From Assumption 4.2.2 and Assumption 4.2.3, because the cost functions are same and the execute times are not changed, the value of the total objective function would not be changed. We can say that it is also optimal. Therefore, by the property of the contradiction, this proposition is true. □

From above two propositions, we can find that it is possible to schedule the tasks in ascending order of their $(t^s_{i,j})_{\min}$ and the result of the optimization with the order will be globally optimal.
The last assumption is about the order of tasks. In most of factories, the work flow is fixed, which means that the tasks or the jobs are flowing in certain direction. Besides, the locations of the robots and work-cells are fixed. So, in many cases, we can easily accept the following assumption:

Assumption 4.2.4. If the task j_1 is ready to start in the robot i before the task j_2 is ready, the task j_1 is also prior to the task j_2 in another robots.

$$\text{If } (t^s_{i_1,j_1})_{\min} \leq (t^s_{i_1,j_2})_{\min}, \text{ then } (t^s_{i_2,j_1})_{\min} \leq (t^s_{i_2,j_2})_{\min},$$
$$\text{for all } i_1, i_2, j_1, j_2 \text{ such that } \mathcal{I}_{i_1,j_1}, \mathcal{I}_{i_1,j_2}, \mathcal{I}_{i_2,j_1}, \mathcal{I}_{i_2,j_2} \text{ exist.}$$

4.3 Algorithm for Task Scheduling

With the assumptions as described in the previous section, we re-formulate the equation (4.1.1) as follows:

$$\begin{aligned} \underset{\omega_j, t^s_{\omega_j,j}, t^f_{\omega_j,j}}{\text{Minimize}} \quad & J = \sum_{j=1}^{m} \mathcal{C}_{\omega_j} (t^f_{\omega_j,j} - t^s_{\omega_j,j}) \\ \text{subject to} \quad & \max_j \left(t^f_{\omega_j,j} \right) \leq t^{\text{complete}} \\ & (t^s_{\omega_j,j})_{\min} \leq t^s_{\omega_j,j} \leq (t^s_{\omega_j,j})_{\max} = (t^s_{\omega_j,j})_{\min} + \Delta_{\omega_j} \\ & t^f_{\omega_j,j} - t^s_{\omega_j,j} \geq d_{\omega_j,j}, \end{aligned} \tag{4.3.5}$$

where t^{complete} is the maximum complete time which is given. With the assumptions, we can solve above problem using our algorithm, much faster and much more practical. Our proposed algorithm is divided into two parts. The first part makes an order of the tasks for the next part, dynamic programming part. The second part performs the optimization which includes the dynamic programming and the trajectory generation.

4.3.1 Task Arranging

Before we perform optimization with trajectory generation algorithm, we need to make an order of the tasks which will be performed in each robot. This part is very simple. From Proposition 4.1 and 4.2, we already know that it is possible to schedule the tasks in ascending order of $(t^s_{i,j})_{\min}$ of tasks and global optimal solution can be found with that order. Therefore, we sort the tasks with the following constraint:

$$\text{If } k_1 < k_2, \text{ then } (t^s_{i,S_{k_1}})_{\min} \leq (t^s_{i,S_{k_2}})_{\min} \quad \text{for all } i \text{ such that } \mathcal{I}_{i,S_{k_1}}, \mathcal{I}_{i,S_{k_2}} \text{ exists,}$$

where S is the result sequence of the tasks, e.g., S_k means k^{th} task. From Assumption 4.2.4, there must be a feasible solution.

4.3.2 Optimization: Dynamic Programming and Trajectory Generation

Using the sequence of the tasks from the previous part, we perform the optimization with dynamic programming method. Dynamic programming is an optimization method which can be applied to problems where the state of the system can be determined by some finite variables and the current decision is effected by only the current state, not the previous decisions. More detailed account can be found in [25].

To solve our problem (4.1.1), we first need to define the state variables representing the system. In our algorithm, we use n variables t_1, ..., t_n. The variable $(t_1, ..., t_n)$ represents the state when the final time of the last motion of robot i is t_i, $i = 1, ...n$. Using these state variables, let's define $J_k^*(t_1, ..., t_n)$ which is the optimal cost when the task $S_1, ..., S_k$ have been performed and the current state of our system is $(t_1, ..., t_n)$. The functional equation of dynamic programming can be written as

$$\begin{aligned} J_{k+1}^*(t_1^f, ..., t_n^f) &= \min_{i,t_i^s} \left[J_k^*(t_1^f, ..., t_i^s, ..., t_n^f) + C_{i,S_{k+1}}(t_i^s, t_i^f) \right], \\ &= \min_{i,t_i^s} \left[J_k^*(t_1^f, ..., t_i^s, ..., t_n^f) + \mathcal{C}_i(t_i^f - t_i^s) \right], \end{aligned} \tag{4.3.6}$$

where $C_{i,S_{k+1}}$ is the optimal energy consumption when the robot i performs the task S_{k+1} from t_i^s to t_i^f. Above equation is to calculate the optimal cost from previous optimal cost J_k^* by adding task S_{k+1}. From (4.3.6), we can find that updating $J_{k+1}^*(t_1^f, ..., t_n^f)$ requires calculating $C_{i,S_{k+1}}(t_i^s, t_i^f)$ $\left(= \mathcal{C}_i(t_i^f - t_i^s)\right)$ for all i and feasible t_i^s. Because the cost function C is also the optimal energy consumption, to get the optimal function value, the optimization is needed. In other words, for our problem, a sub-problem is finding optimal point-to-point trajectory, for example, $C_{i,j}(t^s, t^f)$ $\left(= \mathcal{C}_i(t_i^f - t_i^s)\right)$ is the energy consumption of the optimal trajectory of the robot i for performing the task j from t^s to

t^f. Initial value of J_0^* is zero at $(t_1, ..., t_n) = (0, ..., 0)$, otherwise infinite. Of course, according to problems, additional state variables may be needed and the functional equation will be changed a little.

To solve the problem, it is necessary to quantize the admissible states into finite number of points. The number of points for each state variable is N_t and the total number of grids is $(N_t)^n$. Each state variable is equally divided and the interval is

$$\Delta = \frac{t^{\text{complete}}}{N_t - 1}.$$

If the time complexity for calculating $C_{i,S_{k+1}}(t_i^s, t_i^f)$ $\left(= \mathcal{C}_i(t_i^f - t_i^s)\right)$ once is $O(T)$, then the total time complexity of the task scheduling is $O\left(nm(N_t)^{n+1}T\right)$.

4.3.3 Cost Function Approximation

From the functional equation (4.3.6) and the total time complexity, we can easily infer that the performance of the task scheduling is dominated by the performance of the trajectory generation in terms of computation time and cost. We use our optimal trajectory generation algorithm as described in the previous section. Although our trajectory generation algorithm is computational-efficient, it takes a lot of time to schedule the tasks because calculating the cost function $\mathcal{C}$, optimal trajectory generation, is performed about $nm(N_t)^{(n+1)}$ times. For that reason, we need to calculate the cost function fast. To do that, we approximate the optimal cost function with respect to the final time (task execution time) by calculating the function value and its derivative at a few points (task execution times) for certain task. Because a robot usually performs a repetitive task and we have assumed that a robot can perform similar tasks in the previous section, the approximated optimal cost function for certain task should be useful.

Let's back to Section 3. Using the position, velocity, and acceleration with

normalized time domain which are defined in (3.2.16) and (3.2.17), the torque can be expressed as follows:

$$\tau = \tau\left(\tilde{q}(\tilde{t}), \frac{1}{T}\frac{\mathrm{d}\tilde{q}}{\mathrm{d}\tilde{t}}(\tilde{t}), \frac{1}{T^2}\frac{\mathrm{d}^2\tilde{q}}{\mathrm{d}\tilde{t}^2}(\tilde{t})\right).$$

Solving inverse dynamics with above equation using the closed form of dynamic equations (2.3.5), we can get

$$\tau = f_1(\tilde{t};\theta)\frac{1}{T^2} + f_2(\tilde{t};\theta), \tag{4.3.7}$$

where f_1, f_2 are the nonlinear functions of the normalized time variable $\tilde{t}$ as given by

$$f_1(\tilde{t}) = \mathcal{S}^{\mathrm{T}}G^{\mathrm{T}}\mathcal{J}\left(G\mathcal{S}\frac{\mathrm{d}^2\tilde{q}}{\mathrm{d}\tilde{t}^2} + G\left[\mathrm{ad}_{G\mathcal{S}\frac{\mathrm{d}\tilde{q}}{\mathrm{d}\tilde{t}}}\right]\mathcal{S}\frac{\mathrm{d}\tilde{q}}{\mathrm{d}\tilde{t}}\right) - \mathcal{S}^{\mathrm{T}}G^{\mathrm{T}}\left[\mathrm{ad}_{G\mathcal{S}\frac{\mathrm{d}\tilde{q}}{\mathrm{d}\tilde{t}}}\right]^{\mathrm{T}}\mathcal{J}G\mathcal{S}\frac{\mathrm{d}\tilde{q}}{\mathrm{d}\tilde{t}}$$
$$f_2(\tilde{t}) = \mathcal{S}^{\mathrm{T}}G^{\mathrm{T}}\mathcal{J}GP_0\dot{\mathcal{V}}_0,$$

where $G = G(\tilde{t}) : \mathbb{R} \to \mathbb{R}^{6n\times 6n}$, n is the degrees of freedom of a robot and some parameters θ which do not include the final time T. In our problem, θ includes control points c and some optimization parameters (see Section 4). Note that the base link of the robot is fixed: $V_0 \in se(3)$ is zero. There are two important facts. The first one is that, when all variables and parameters are fixed except for the final time T, τ is the linear combination of T^{-2} and T^0. Second, if $\tilde{q}(\tilde{t})$, $\mathrm{d}\tilde{q}/\mathrm{d}\tilde{t}(\tilde{t})$, and $\mathrm{d}^2\tilde{q}/\mathrm{d}\tilde{t}^2(\tilde{t})$ are bounded, $f_1(\tilde{t};\theta)$ and $f_2(\tilde{t};\theta)$ are also bounded, because all elements of the matrices in (2.3.5) are bounded. When there are only two waypoints, substituting above result into the objective function (3.4.21) with effort function and energy loss function, then we can get

$$\begin{aligned}\mathcal{C}_{\mathrm{effort}}(\theta,T) &= g_1(\theta)\frac{1}{T^3} + g_2(\theta)\frac{1}{T} + g_3(\theta)T \\ \mathcal{C}_{\mathrm{energyloss}}(\theta,T) &= g'_1(\theta)\frac{1}{T^3} + g'_2(\theta)\frac{1}{T^2} + g'_3(\theta)\frac{1}{T} + g'_4(\theta) + g'_5(\theta)T,\end{aligned} \tag{4.3.8}$$

where g and g' are the functions of parameter θ which does not include the final time T. Because g and g' come from f_1 and f_2 and the sum of weights

of Gaussian Quadrature in (3.4.21) is 1, we can easily find that g and g' are also bounded when $\tilde{q}(\tilde{t})$, $\mathrm{d}\tilde{q}/\mathrm{d}\tilde{t}(\tilde{t})$, and $\mathrm{d}^2\tilde{q}/\mathrm{d}\tilde{t}^2(\tilde{t})$ are bounded. For our problem, those are bounded because the control points of B-splines are limited by the position constraints. So, in this paper, we can think that g and g' are always bounded. Note that we are optimizing above objective functions with some constraints for trajectory generation and optimization parameters are the parameter θ and final time T which identify the trajectory of the robot.
Now, let's think what happened to the result of the optimization when the task execution time T is enough large. First, the constraints for velocities and accelerations of the joints are inactive. The reason can be easily found from the fact that the knots of B-spline we used are fixed and the control points are bounded by the position constraints. Next, if the position constraints are weak (note that we are focusing on the robot which have only revolute joints, so the joint values are angles) and an object which the robot picks is not much heavy, the position constraints and torque constraints are also inactive. Therefore, when the task execution time T is enough large, all constraints are inactive. If an objective function is $\mathcal{C}$ and T is given, the optimal parameter $\theta^* = \theta^*(T)$ satisfies the following:

$$\left.\frac{\partial \mathcal{C}}{\partial \theta}\right|_{\theta^*,T} = 0. \tag{4.3.9}$$

If $\mathcal{C}^*(T) = \mathcal{C}(\theta^*, T)$, then

$$\begin{aligned}\left.\frac{\mathrm{d}\mathcal{C}^*}{\mathrm{d}T}\right|_T &= \left.\frac{\partial \mathcal{C}}{\partial T}\right|_{\theta^*,T} + \left.\frac{\partial \mathcal{C}}{\partial \theta}\right|_{\theta^*,T}\frac{\mathrm{d}\theta}{\mathrm{d}T} \\ &= \left.\frac{\partial \mathcal{C}}{\partial T}\right|_{\theta^*,T}.\end{aligned}$$

Finally, with above results, we can easily prove the following proposition.

Proposition 4.3. *If an optimal parameter is $\theta^* = \theta^*(T)$ when an objective*

function $\mathcal{C}$ and constraints are given as (3.4.21) *and* (4.3.8)*, the optimal parameter will converge when the task execution time T approaches infinity.*

Proof. Let's prove this proposition with an effort function. With similar procedure, it can be proven when the objective function is given by an energy loss function.

From (4.3.8), the partial derivative of an objective function can be written as

$$\frac{\partial \mathcal{C}}{\partial \theta} = \frac{\partial g_1}{\partial \theta}\frac{1}{T^3} + \frac{\partial g_2}{\partial \theta}\frac{1}{T} + \frac{\partial g_3}{\partial \theta}T.$$

Then, when $\theta = \theta^*(T)$, we can get

$$\begin{aligned}\frac{\partial \mathcal{C}}{\partial \theta} &= \frac{\partial g_1}{\partial \theta}\frac{1}{T^3} + \frac{\partial g_2}{\partial \theta}\frac{1}{T} + \frac{\partial g_3}{\partial \theta}T \\ &= 0 \\ \frac{\partial g_3}{\partial \theta} &= -\frac{1}{T^4}\left(\frac{\partial g_1}{\partial \theta} + \frac{\partial g_2}{\partial \theta}T^2\right).\end{aligned}$$

As the control points in the optimization parameter θ are bounded by the position constraints (see (3.4.21)) and $\tilde{q}$, $\mathrm{d}\tilde{q}/\mathrm{d}\tilde{t}$, and $\mathrm{d}^2\tilde{q}/\mathrm{d}\tilde{t}^2$ are B-splines defined by those control points, the partial derivatives of $\tilde{q}$, $\mathrm{d}\tilde{q}/\mathrm{d}\tilde{t}$, and $\mathrm{d}^2\tilde{q}/\mathrm{d}\tilde{t}^2$ with respect to the control points, θ, are bounded (see (3.2.19)). So, we can easily find that $\partial f_1/\partial\theta$ and $\partial f_2/\partial\theta$ are bounded, where f_1 and f_2 can be found in (4.3.7). As g_1, g_2, and g_3 are functions of f_1 and f_2, $\partial g_1/\partial\theta$, $\partial g_2/\partial\theta$, and $\partial g_3/\partial\theta$ are also bounded. So, the limit of $\partial g_3/\partial\theta$, as T approaches infinity, can be written as

$$\lim_{T\to\infty} \left.\frac{\partial g_3}{\partial \theta}\right|_{\theta^*(T)} = 0.$$

As θ^* is an optimal parameter which minimizes $\mathcal{C}$ when T is given, we can say that, as T goes infinity, the optimal parameter θ^* approaches the parameter which minimizes g_3 (note that g_1, g_2, and g_3 are bounded). Let's θ^{opt} be the optimal parameter that minimizes g_3. Because the joint position, velocity, and

acceleration are limited, θ^* must exist. So we can write the following equation:

$$\lim_{T\to\infty} \theta^*(T) = \theta^{\text{opt}}.$$

□

What we want to do from now on is to determine whether assuming that the optimal parameter is constant is reasonable or not. From Proposition 4.3, we know that the optimal parameter will converge, as the task execution time T goes to infinity. However the problem is that the task execution time we are interested in is not enough large. So, we need to check how fast the optimal parameter converges and determine whether this assumption is reasonable in our problem or not. From (4.3.9), the Talyor series of the partial derivative of the objective function with respect to the parameter θ at a critical point $(\theta^*(T), T)$ for $\Delta\theta^*$ and ΔT can be expressed as follows:

$$\left.\frac{\partial \mathcal{C}}{\partial \theta}\right|_{\theta^*+\Delta\theta^*,T+\Delta T} = 0 = \left.\frac{\partial \mathcal{C}}{\partial \theta}\right|_{\theta^*,T} + \left.\frac{\partial^2 \mathcal{C}}{\partial \theta^2}\right|_{\theta^*,T} \Delta\theta^* + \left.\frac{\partial^2 \mathcal{C}}{\partial \theta \partial T}\right|_{\theta^*,T} \Delta T + \text{high order}.$$

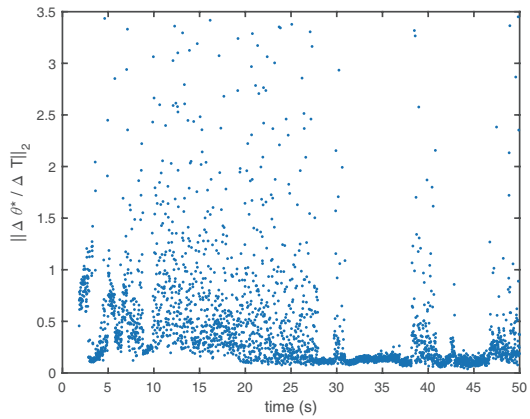
Ignoring the high order term, we can get

$$\frac{\Delta\theta^*}{\Delta T} \approx \left(\left.\frac{\partial^2 \mathcal{C}}{\partial \theta^2}\right|_{\theta^*,T}\right)^{-1} \left.\frac{\partial^2 \mathcal{C}}{\partial \theta \partial T}\right|_{\theta^*,T}.$$

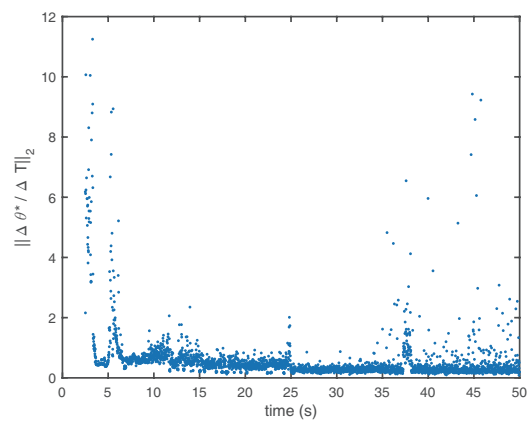
Figure 4.3 shows how much the optimal parameter changes as the task execution time T goes up. We solve the optimization problem which is defined in Section 3 to get θ^* for each task execution time (final time) and calculate the following value:

$$\left\|\frac{\Delta\theta^*}{\Delta T}\right\|_2 \approx \left\| \left(\left.\frac{\partial^2 \mathcal{C}}{\partial \theta^2}\right|_{\theta^*,T}\right)^{-1} \left.\frac{\partial^2 \mathcal{C}}{\partial \theta \partial T}\right|_{\theta^*,T} \right\|_2,$$

where $\partial^2\mathcal{C}/\partial\theta^2$ and $\partial^2\mathcal{C}/\partial\theta\partial T$ are calculated numerically. Of course, the result depends on some internal factors, e.g., constraints and parameters of



(a) Without payload



(b) With payload (20kg)

Figure 4.3: The change of the optimal parameter with respect to the change of the task execution time (the objective function is given by the energy loss function)

kinematics and dynamics, and some external factors, e.g., a payload and external forces. However we can heuristically find that, in the range we are interested in, the change of the optimal parameter is enough small and it converges quickly enough to be used as constant for approximating the optimal

cost function which will be described in the follows.

We show that it is possible to assume that the optimal parameter is constant. Then we can get the fact that $g_i(\theta^*)$ and $g'_j(\theta^*)$ $(i = 1, 2, 3, j = 1, ..., 5)$ are also constant. So, with $\theta^*(T)$, we can rewrite (4.3.8) as

$$\begin{aligned} \mathcal{C}^*_{\text{effort}}(T) &\approx G_1 \frac{1}{T^3} + G_2 \frac{1}{T} + G_3 T \\ \mathcal{C}^*_{\text{energyloss}}(T) &\approx G'_1 \frac{1}{T^3} + G'_2 \frac{1}{T^2} + G'_3 \frac{1}{T} + G'_4 + G'_5 T, \end{aligned} \tag{4.3.10}$$

where $G_i \in \mathbb{R}$ and $G'_j \in \mathbb{R}$ $(i = 1, 2, 3, j = 1, ..., 5)$ are constant. Using only three function values and their derivatives, we can easily calculate the coefficients G $(= (G_1, G_2, G_3))$ and G' $(= (G'_1, ..., G'_5))$. Fortunately, we already have energy-optimal trajectory generation algorithm, so we can get the function value $\mathcal{C}^*(T)$ and we can also calculate its partial derivative analytically. We use least square method for minimizing the least square error:

$$\underset{G}{\text{Minimize}} \sum_i \left(\omega \left| \mathcal{C}^*(T_i; G) - \mathcal{C}^*_{\text{traj}}(T_i) \right|^2 + (1 - \omega) \left| \frac{\mathrm{d}\mathcal{C}^*}{\mathrm{d}T}(T_i; G) - \frac{\partial \mathcal{C}^*_{\text{traj}}}{\partial T}(T_i) \right|^2 \right),$$

where ω is a weight factor and $\mathcal{C}^*_{\text{traj}}$ is a function of T, e.g., $\mathcal{C}^*_{\text{traj}}(T)$ is the result of our trajectory generation when T is given, and $\partial \mathcal{C}^*_{\text{traj}} / \partial T$ is its partial derivative with respect to T. To obtain the coefficients, we use function values and corresponding derivatives at four points:

$$\begin{aligned} &T_1 : \text{minimum final time } \left(T_1 = \underset{T}{\text{argmin}} \int_0^T \mathrm{d}t \right) \\ &T_2 : \text{minimum objective function } \left(\frac{\mathrm{d}\mathcal{C}^*}{\mathrm{d}T}(T_2) = 0 \right) \\ &T_3 : \frac{T_2 + T_4}{2} \\ &T_4 : \text{enough long time } \approx t^{complete}. \end{aligned}$$

Therefore, for energy loss case, the optimal coefficients are calculated as the following equation:

$$G' = \left\{ \omega (A^{\mathrm{T}} A) + (1 - \omega)(A'^{\mathrm{T}} A') \right\}^+ \left\{ \omega A^{\mathrm{T}} b + (1 - \omega) A'^{\mathrm{T}} b' \right\},$$

where $\{\cdot\}^+$ means pseudo inverse, $\omega = 0.05 \sim 0.1$, $G' = [G'_1, ..., G'_5]^{\mathrm{T}}$ and

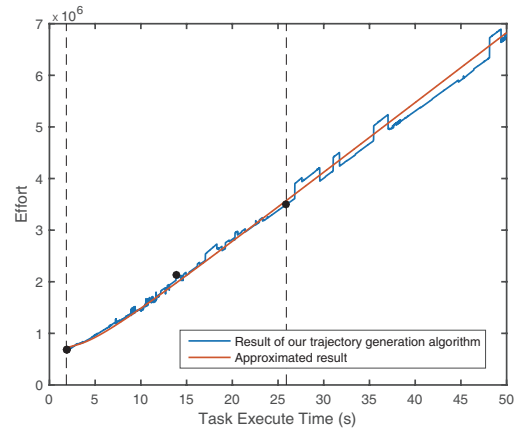
$$A = \begin{bmatrix} \frac{1}{T_1^3} & \frac{1}{T_1^2} & \frac{1}{T_1} & 1 & T_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{T_4^3} & \frac{1}{T_4^2} & \frac{1}{T_4} & 1 & T_4 \end{bmatrix} \in \mathbb{R}^{4\times 5}, \qquad b = \begin{bmatrix} J^*_{\mathrm{traj}}(T_1) \\ \vdots \\ J^*_{\mathrm{traj}}(T_4) \end{bmatrix} \in \mathbb{R}^4$$

$$A' = \begin{bmatrix} -3\frac{1}{T_1^4} & -2\frac{1}{T_1^3} & -\frac{1}{T_1^2} & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -3\frac{1}{T_4^4} & -2\frac{1}{T_4^3} & -\frac{1}{T_4^2} & 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 5}, \qquad b' = \begin{bmatrix} \frac{\partial J^*_{\mathrm{traj}}}{\partial T}(T_1) \\ \vdots \\ \frac{\partial J^*_{\mathrm{traj}}}{\partial T}(T_4) \end{bmatrix} \in \mathbb{R}^4.$$

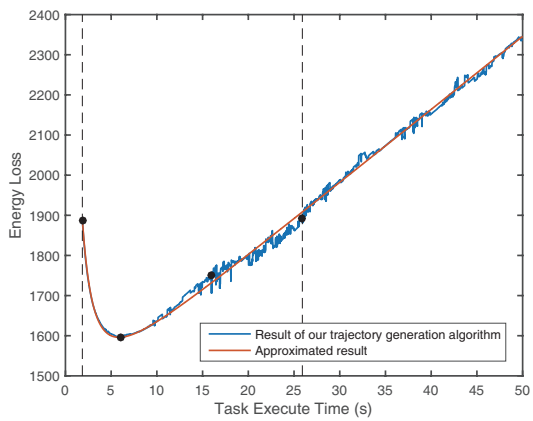
Figure 4.4 represents the result of the approximating method. To verify our approximating method, we first perform the optimization for each final time using our trajectory generation algorithm. Blue line is for the result of ours with given final time (task execute time). It shows that the blue line is not smooth. The reason may be that there are local minimums (we optimize the trajectories from same initial guess). Red line represents the approximated function using the function values at black points and their derivatives. Dash lines represent T_1 and T_4. To evaluate the approximating functions, we also perform the optimization after T_4 (see blue lines), and we draw the approximated functions (see red lines) and then we compare the tendencies of two graphs. From the result, we can argue that the functions are well approximated. For the case with a waypoint, total optimal energy consumption can be formulated as

$$\mathcal{C}^*(T) = \min_{T'} \left[\mathcal{C}^*_{\text{section 1}}(T') + \mathcal{C}^*_{\text{section 2}}(T - T') \right],$$

where $\mathcal{C}^*_{\text{sector 1}}$ and $\mathcal{C}^*_{\text{sector 2}}$ are the optimal energy consumption for sector 1 and 2, respectively. Of course, because there are so many options to set waypoints, it does not make sense to solve above equation for every cases in general. So, we just assume that the optimal time T' only depends on the total time T,



(a) Optimal effort for given task execute time



(b) Optimal energy loss for given task execute time

Figure 4.4: Approximated function to optimal energy consumption with respect to task execute times

which can express as

$$T' = \gamma T, \quad \gamma \in [0, 1].$$

Then, we can say that, even if there is a waypoint, approximation (4.3.10) is valid. Now, for each pair of a task and robot, we can get optimal energy consumption with respect to any task execution time in $O(1)$ by performing

optimization only four times. Although the optimized result and approximated function values are not exactly same, in fact, it does not matter. The more important things are which time the minimum energy consumption happens and how much the energy consumption increases as the task execute time goes up. From that point of view, our models and assumptions we applied are proper.

4.4 Example: Pick and Place Motion

To evaluate our algorithm, we simulate pick and place motion with two robots and two conveyor belts as shown in Figure 4.5. Several objects are sitting on

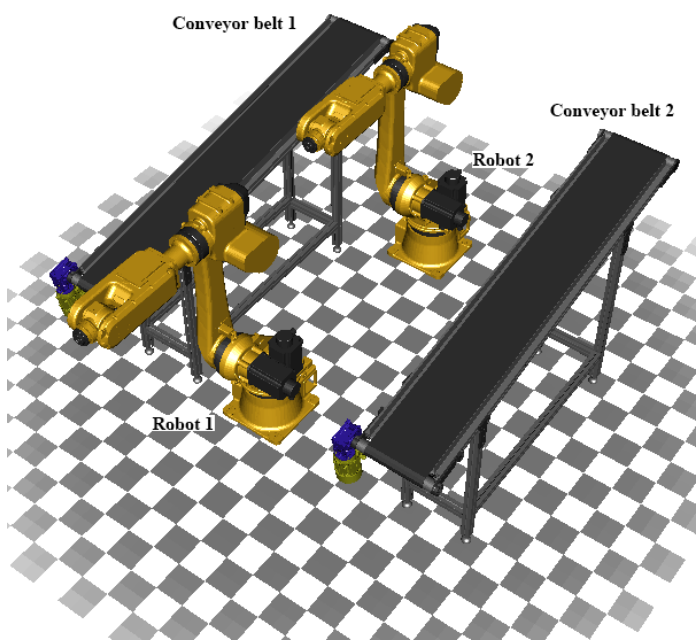
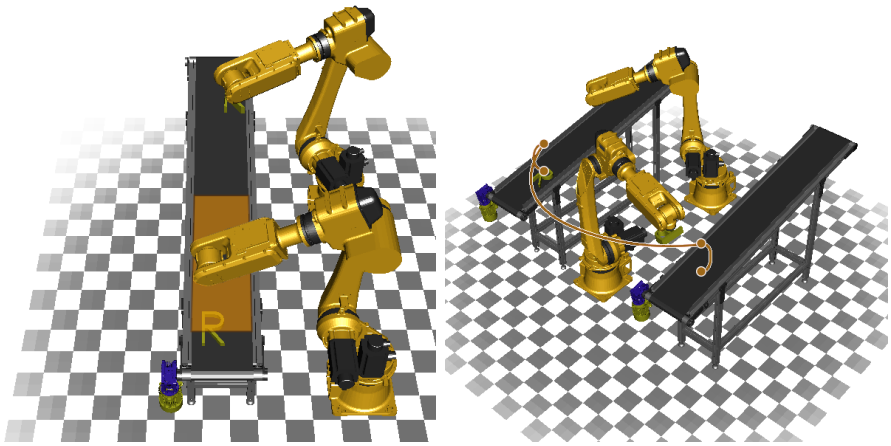


Figure 4.5: Simulation setting for pick and place motion

the moving conveyor belt 1. The task is to pick and place those objects from



(a) Available picking area for Robot 1 (orange area)

(b) Example of waypoints for pick and place motion (orange points and line)

Figure 4.6: Waypoint conditions or task constraints for pick and place motion

conveyor belt 1 to conveyor belt 2. As you can see in Figure 4.6(a), each robot has an available place for picking an object. In other words, an object outside that area cannot be picked by the corresponding robot. For each robot, the place where the robot puts the object is defined. The pick and place motion is defined by not only picking point and placing point but also two waypoints as shown in Figure 4.6(b). The two waypoints are set near by the picking and placing points, respectively. In addition, times, velocities, accelerations of the robot at the waypoints are free. The task is defined as the motion for picking and placing. The task starts at the time when the robot begins to move for picking the object and finishes at the time when the object is put on the conveyor belt 2.

In our example, there are 6 objects. At the beginning, the objects are put on the conveyor belt 1 in order. We assume that each of objects is $20kg$ and its inertia is defined as

$$20 \times I^{6\times 6},$$

where $I^{6\times6}$ is a 6 by 6 identity matrix. As we mentioned before, because problems are various, corresponding state variables and functional equations are also varying. For our example, we use additional state variables for distinguishing zero position and switching position which occurs between two tasks. To find out effectiveness of our algorithm, we compare the result of our algorithm to the result from Reflexxes library. To get the trajectories and calculate the energy consumption, the picking time and the position of the waypoints are set as the result of our algorithm. The trajectories from Reflexxes are to pick and place each object as fast as the robot can, which is a method that is mainly used in current factories. Table 4.1 shows how much energy is used by the trajectories from our algorithm and Reflexxes. Here, for the Reflexes' result, we did not consider the energy consumption which the robots consume during the idle time. So we denote it as α. Of course, α must be greater than zero. Note that the objective value in Table 4.1 does not come from the approximated cost functions. The trajectories from both our algorithm and Reflexxes can be used for picking and placing all objects under given constraints. However our algorithm uses much less energy consumption than Reflexxes for performing all tasks.

Table 4.1: Result of scheduling optimization

Objective Function Type	Our algorithm	Reflexxes
Energy loss$^{(1)}$	16677.55	19135.82 $+\alpha$
Energy loss$^{(2)}$	18758.2	19450.34 $+\alpha$

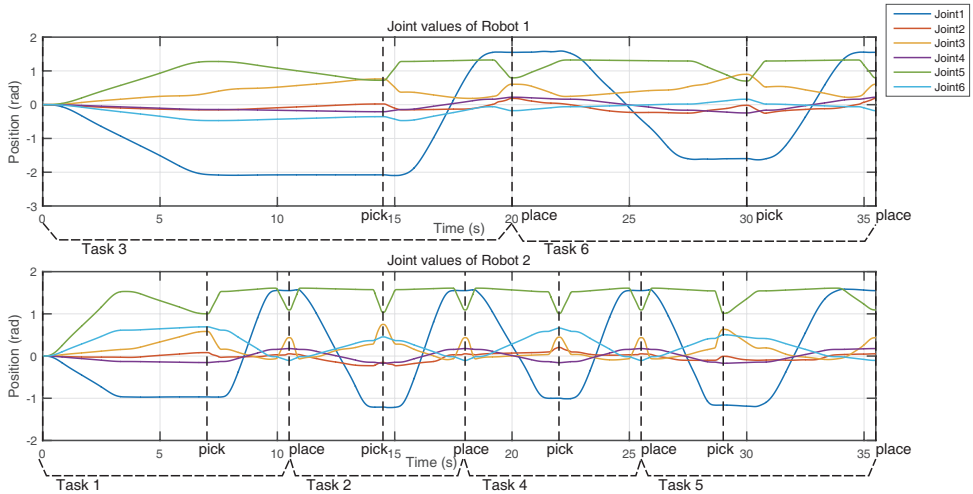


Figure 4.7: Joint trajectory of each robot for task scheduling (for the energy $\text{loss}^{(1)}$ case)

5

Conclusion

This thesis has proposed an energy-optimal task scheduling algorithm with a point-to-point trajectory generation method under kinematic and dynamic constraints and various types of boundary conditions. We have considered all things from a low level, e.g., a hardware specification, to a high level, e.g., task information and a scheduling problem, and also we have integrated them. A set of equations and methods about the energy consumption has been introduced or derived in this thesis. Using that, the joint torques for the objective functions have been calculated and the trajectory generation and task scheduling have been performed. Those results let us make our algorithms computationally efficient.

Focusing on the integration of the task scheduling and the point-to-point trajectory generation with respect to energy optimality, the thesis makes two contributions:

- **Energy-optimal trajectory generation**: We have proposed an energy-optimal point-to-point trajectory generation algorithm for multiple waypoints and various types of boundary conditions with free execution

times using B-spline curves, normalized time domains, and the Gaussian quadrature method. To make our algorithm computationally efficient, we have calculated the analytic gradients of the objective functions by a recursive differential inverse dynamics. From the case studies have been shown in this thesis, we have found that the algorithm we have proposed can be used in much more cases compared to preceding algorithms.

- **Task scheduling of energy-optimal trajectories**: We have also proposed the energy-optimal task scheduling algorithm using the energy-optimal trajectory generation algorithm as we have described in this thesis. We have first defined a general problem and applied several assumptions to make it more practical. We have used a dynamic programming method for optimization and we have also used an optimal energy consumption measurement which is approximated for the computational efficiency. An example, a pick-and-place motion, has shown that we have used an idle time which exists when it is performed by a time-optimal trajectory for reducing the energy consumption. In addition, the approximation of the optimal energy consumption measurement has dramatically reduced the computation time of the algorithm.

This thesis can be a cornerstone of the energy-optimal task scheduling in the industrial field. However, it still has a long way to go before being used in practice, e.g., a collision-free path and an on-line task scheduling. In fact, because requirements of factories are varying, it is impossible to consider all of them and generalize a problem. Therefore, for the energy-optimal task scheduling, collecting the components which are needed for their problem and integrating those components are main issues.

Although this thesis has considered a very small part of the energy-optimal task scheduling, it has shown the possibility of the effect of the energy-optimal

task scheduling and the necessity of the integration of the energy-optimal task scheduling and the point-to-point trajectory generation. Through those results, it is hoped that this thesis can inspire other researchers and help them to research the energy-optimal task scheduling for being used in practice.

Bibliography

[1] C. Hansen, J. Kotlarski, D.Meike, and T. Ortmaier. Enhanced approach for energy-efficient trajectory generation of industrial robots. In *Proceedings of 2012 8th IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1–7. IEEE, Aug 2012.

[2] S. Ntemiri (European Environment Agency). Total greenhouse gas emission trends and projections. *http://www.eea.europa.eu/data-and-maps/indicators/greenhouse-gas-emission-trends-5/assessment-1*.

[3] European Commission. Energy efficiency, europe 2020 targets. *http://ec.europa.eu/europe2020/targets/eu-targets*.

[4] D. Meike and L. Ribickis. Energy efficient use of robotics in the automobile industry. In *Proceedings of 2011 15th IEEE International Conference on Advanced Robotics (ICAR)*, pages 507 – 511. IEEE, Jun 2011.

[5] INNOVATIONSEEDS. Eco-point incentive program in japan. *http://www.innovationseeds.eu/Policy-Library/Core-Articles/Eco-Point-Incentive-Program-In-Japan.kl*.

[6] M. Song, T. J. Tarn, and N. Xi. Integration of task scheduling, action planning, and control in robotic manufacturing systems. In *Proceedings of the IEEE*, volume 88, pages 1097–1107. IEEE, Jul 2000.

[7] H. Diken. Energy efficient sinusoidal path planning of robot manipulators. *Mechanism and Machine Theory*, 29(6):785–792, Aug 1994.

[8] K. Paes, W. Dewulf, K. V. Elst, K. Kellens, and P. Slaets. Energy efficient trajectories for an industrial abb robot. In *Procedia CIRP*, volume 15, pages 105–110. 21st CIRP Conference on Life Cycle Engineering, Jun 2014.

[9] B. J. Martin and J. E. Bobrow. Minimum-effort motions for open-chain manipulators with task-dependent end-effector constraints. *International Journal of Robotics Research*, 18(2):213–224, Feb 1999.

[10] C. Y. E. Wang, W. K. Timoszyk, and J. E. Bobrow. Payload maximization for open chained manipulators: finding weightlifting motions for a puma 762 robot. *IEEE Transactions on Robotics and Automation*, 17(2):218–224, Apr 2001.

[11] Joonggon Kim. *Dynamics Based Motion Optimization for Skeleton Driven Deformable Body System.* PhD thesis, Seoul National University, 2007.

[12] O. Wigström, B. Lennartson, A. Vergnano, and C. Breitholtz. High-level scheduling of energy optimal trajectories. *IEEE Transactions on Automation Science and Engineering*, 10(1):57–64, Jan 2013.

[13] M. Pellicciari, G. Berselli, F. Leali, and A. Vergnano. A method for reducing the energy consumption of pick-and-place industrial robots. *Mechatronics*, 23(3):326–334, Apr 2013.

[14] X. J. Wang, C. Y. Zhang, L. Gao, and P. G. Li. A survey and future trend of study on multi-objective scheduling. In *Proceedings of the 4th International Conference on Natural Computation*, pages 382–391. ICNC, Oct 2008.

[15] A. Vergnano, C. Thorstensson, B. Lennartson, P. Falkman, M. Pellicciari, F. Leali, and S. Biller. Modeling and optimization of energy consumption in cooperative multi-robot systems. *IEEE Transactions on Automation Science and Engineering*, 9(2):423–428, Jan 2012.

[16] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[17] F. C. Park, J. E. Bobrow, and S. R. Ploen. A lie group formulation of robot dynamics. *International Journal of Robotics Research*, 14(6):609–618, Dec 1995.

[18] S. R. Ploen and F. C. Park. Coordinate-invariant algorithms for robot dynamics. *IEEE Transactions on Robotics and Automation*, 15(6):1130–1135, Dec 1999.

[19] S. R. Ploen. *Geometric Algorithms for the Dynamics and Control of Multibody Systems*. PhD thesis, University of California, Irvine, 1997.

[20] C. Hansen, J. Kotlarski, and T. Ortmaier. Experimental validation of advanced minimum energy robot trajectory optimization. In *Proceedings of 2013 16th IEEE International Conference on Advanced Robotics (ICAR)*, pages 1–8. IEEE, Nov 2013.

[21] A. Gasparetto and V. Zanotto. Optimal trajectory planning for industrial robots. *Advances in Engineering Software*, 41(4):548–556, Apr 2010.

[22] C. de Boor. *A practical guide to splines*. New York: Springer-Verlag, 1978.

[23] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *NUMERICAL RECIPES: The Art of Scientific Computing.* Cambridge University Press, third edition, 2007.

[24] S. G. Johnson. The nlopt nonlinear-optimization package. *http://ab-initio.mit.edu/wiki/index.php/NLopt.*

[25] D. E. Kirk. *Optimal Control Theory: An Introduction.* Prentice Hall, Englewood Cliffs New Jersey, 1970.

국문초록

이 논문은 기구학적, 동역학적 조건에서 에너지 효율을 높이는 작업 스케줄링에 대해 다루고 있다. 작업 스케줄링에서 에너지 최적화를 수행 위해서는 로봇 매니퓰레이터의 최적 경로 생성이 같이 수행 되어져야하기 때문에, 작업 스케줄링과 최적 경로 생성의 통합 이슈는 이 논문에서 중요하게 다루어진다. 먼저 우리는 지나가야하는 경로점이 주어졌을 때 최적 에너지 경로 생성 알고리즘을 제시한다. 최적화 문제는 여러 개의 경로 점이 주어지거나 다양한 경계 조건들이 있거나 작업 수행 시간을 최적화해야 하는 경우를 다룰 수 있도록 정의되었다. 모든 경로들은 C-공간(조인트 공간)에서 B-spline으로 매개화 되었으며 목적함수들은 재귀적인 역 다이나믹스 방법으로 계산된다. 우리는 최적화 알고리즘의 계산 효율을 위해 해석적인 미분을 이용한다. 또한 적분을 위해 가우시안 구적법을 사용한다. 최적 에너지 경로 생성 알고리즘의 성능을 평가하기 위해 우리는 몇 가지 상황에서 경로를 생성해보고 그 결과를 분석한다. 또한 우리는 이 논문에서 동적 계획법을 이용한 작업 스케줄링 알고리즘을 제시한다. 우리는 먼저 몇 가지 가정을 통해 현실적인 문제 정의를 내린다. 알고리즘은 작업마다 최적의 로봇을 결정하고 언제 작업을 시작하면 좋을지 판단하며 최적의 작업 수행 시간을 찾는다. 얼마나 에너지 소비를 했는지는 이 논문에서 제시한 경로 최적화 알고리즘에 의해 계산되며 우리는 계산 량을 줄이기 위해 에너지 소비 함수를 근사하여 사용한다.

주요어: 개연쇄 로봇, 에너지 최적확, 두 지점 경로 계획, 베이스 링크 최적화, 태스크 스케줄링, B 스플라인

학번: 2014-21857