



NTNU – Trondheim
Norwegian University of
Science and Technology

Time Optimal Motion Planning and Motion Control for Industrial Manipulators

Marius Nordheim Røv

Master of Science in Cybernetics and Robotics

Submission date: May 2014

Supervisor: Anton Shiriaev, ITK

Co-supervisor: Stepan Pchelkin, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Title: Time Optimal Motion Planning and Motion Control
for Industrial Manipulators

Student: Marius Nordheim Røv

Problem description:

Planning time optimal trajectories for industrial robot manipulators is one of the demanding subjects in robotics since even a small increase in productivity can generate a substantial profit in production. The task assumes planning time-optimal trajectories for given scenarios for the robot ABB IRB140 (circular and straight line motion of the TCP) and validating/analyzing the results when applying a control strategy. In particular, we are interested in the possibility of connecting two (or more) path segments, each with their own assigned velocity profile.

The following steps in investigation are planned:

- Developing dynamical model for the robot ABB IRB140.
- Analysis of geometric and velocity constraints.
- Applying methods for path generation and velocity assignment (motion planning).
- Investigating velocity assignments for connection points between path segments.
- Validation/analysis of the results with applied motion control.

The thesis should be organized according to existing guidelines, and delivered at the Department of Engineering Cybernetics within the 2nd of June.

Supervisor: Professor Anton Shiriaev, ITK

Co-Supervisors: Postdoctoral Fellow Leonid Paramonov, ITK

PhD Candidate Stepan Pchelkin, ITK

Abstract

The ever growing need to make industry more efficient and safe has introduced robots as potential replacements for manual labour. Even minor increases in productivity can generate substantial profit in production, which makes time optimal control one of the demanding subjects in robotics. In the development of robotic systems, two major aspects can be distinguished: motion planning and motion control. The purpose of motion planning is to generate a set of desired trajectories (profiles) for each actuator of the system on a path. The goal of the motion controller is to achieve accurate and robust tracking of the given profiles. In this thesis we have investigated these two aspects with regards to time optimal motion for paths that are exactly known in advance.

We assigned velocity profiles using a method called path constrained trajectory planning on a set of case paths. The resulting motion planning profiles were simulated for an industrial robot manipulator, ABB IRB140. The dynamics of the robot were modeled, and a state-of-the-art orbital stabilization based control structure was implemented. The motion planning and simulations with motion control resulted in faster periods of motion than what could be obtained with the commercial planner from the robot producer company, ABB.

In literature, velocity profiles are usually given with velocity set to zero at the initial and final points for given paths. When connecting two path segments into new and longer paths, it would be desirable to keep some velocity through the connection point, to achieve near time optimal motion periods. We have combined a set of paths and analyzed the results from simulations when applying various velocity assignments at the connection points. When connecting paths that are well geometrically conditioned, we found that we can set boundary level velocity through the connection point and still track the desired trajectories with high accuracy. This was exemplified with a horizontal circle connected to a vertical circle, which performed with a maximum deviation of 0.005 radians from its desired trajectory. For sharp corner connection points, tracking fails. A possible solution was found by implementing a transition phase to round off corners. This was exemplified for a connection point with a 90 degree corner using boundary level velocity profiles, and it was shown how to tune the transition phase to obtain better results.

Sammendrag (Norwegian)

Det alltid økende behovet for å gjøre industriproduksjon mer trygt og effektivt har lansert roboter som potensielle erstattere for manuell arbeidskraft. Selv små forbedringer i produktiviteten kan generere betydelig profitt i produksjonen, noe som gjør tidsoptimalisering til et av de mest etterspurte tema innen robotikk. I utviklingen av robotsystemer, kan vi skille mellom to aspekter: bevegelsesplanlegging og bevegelsesregulering. Formålet i bevegelsesplanleggingen er å generere et sett av ønskede hastighetsprofiler for hver aktuator i systemet på en gitt bane. Målet til bevegelsesregulatoren er å oppnå nøyaktig og robust sporing av de gitte profilene. I denne avhandlingen har vi undersøkt disse to aspektene med tanke på tidsoptimal bevegelse når banene er forhåndskjente.

Vi har tildelt hastighetsprofiler ved bruk av en metode kalt 'path-constrained trajectory planning' på et sett av baner. De resulterende hastighetsprofilene ble simulert for en industriell robot-manipulator, ABB IRB140. Dynamikkene til roboten ble modellert, og en state-of-the-art regulatorstruktur baser på orbital stabilisering ble implementert. Bevegelsesplanleggingen og simuleringer med bevegelsesregulator resulterte i raskere tidsperioder for bevegelsene enn hva som kunne oppnås med den kommersielle planleggeren fra robot-produsent-selskapet, ABB.

I litteraturen er som regel hastighetsprofiler gitt verdien null i start- og sluttpunktene for gitte baner. Når man tilkobler to bane-segmenter til nye og lengre baner, vil det være ønskelig å beholde hastighet gjennom tilkoblingspunktet for å oppnå en nær tidsoptimal bevegelsestid. Vi har kombinert et sett baner, og analysert resultatet fra simuleringer hvor forskjellige hastighetsprofiler er tilordnet i tilkoblingspunktet. Når man tilkobler baner som er godt geometrisk tilpasset, fant vi at man kunne sette hastigheten til grenseverdier og fortsatt oppnå høy nøyaktighet. Dette ble vist med en horisontal sirkel tilkoblet en vertikal sirkel, noe som ga et maksimalavvik på 0.005 radianer fra ønsket baneprofil. For tilkoblingspunkter med skarpe hjørner, feiler banesporingen. En mulig løsning ble funnet ved å implementere en overgangsfase for å avrunde hjørner. Dette ble eksemplifisert for et 90 graders hjørne med hastighetsprofil satt til grenseverdier, og det ble vist hvordan man kan innstille overgangsfasen for å oppnå bedre resultater.

Preface

This master thesis was given in the spring of 2014 by the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU) as part of the M.Sc. program in engineering cybernetics. The work is a continuation of the project work from the fall of 2013. My supervisor has been Professor Anton Shiriaev from the Department of Engineering Cybernetics, NTNU. My Co-Supervisors have been Postdoctoral Fellow Leonid Paramonov and PhD Candidate Stepan Pchelkin, both from the Department of Engineering Cybernetics, NTNU.

I would like to thank my supervisor, Prof. Anton Shiriaev, for his guidance through these interesting topics over the last year. I would also like to thank the Co-Supervisors for providing me with valuable help and answers during the project work leading up to the master thesis. Lastly, I would like to thank my family. I am so grateful for your support during my years at NTNU.

Trondheim, May 2014
Marius Nordheim Røv

Contents

List of Figures	xi
List of Tables	xvii
1 Introduction	1
1.1 Motivation and Introduction to Industrial Robot Manipulators	1
1.1.1 The IRB140 Robot Manipulator	2
1.2 Scope and Emphasis	2
1.3 Outline of Report	3
2 Concepts and Theories	5
2.1 Robot Manipulator Kinematics	5
2.1.1 The Denavit-Hartenburg Convention	5
2.1.2 Forward Kinematics	7
2.1.3 Inverse Kinematics	7
2.2 Robot Manipulator Dynamics	8
2.2.1 The Euler Lagrange Equations	8
2.2.2 Holonomic Constraints	10
2.2.3 The Recursive Newton-Euler Formulation	10
2.3 Motion Planning Terminology	13
2.4 Motion Control Theory	15
3 Mathematical Modeling of the IRB140 Manipulator	17
3.1 Kinematics	17
3.1.1 Forward Kinematics	17
3.1.2 Inverse Kinematics	20
3.1.3 Velocity Kinematics	21
3.2 Dynamics	22
3.2.1 Parameter Estimation	22
3.2.2 Finding Vectors and Mass Centers	22
3.2.3 The Inertia Tensor	23
3.2.4 The Newton-Euler Formulation	24

3.3	Discussion	26
4	Motion Planning	27
4.1	Path Planning	27
4.2	Path-Constrained Trajectory Planning	28
4.2.1	Time Optimization	29
4.2.2	Virtual Holonomic Constraints	30
4.2.3	Finding Velocity Profiles	30
4.3	Motion Planning on Case Paths	33
4.3.1	Case 1: Horizontal Circular Motion	33
4.3.2	Case 2: Vertical Circular Motion	39
4.3.3	Case 3: Straight Line Motion	43
4.4	Discussion	47
5	Motion Control	49
5.1	Orbital Stabilization	49
5.2	Motion Control Simulations on Case Trajectories	50
5.2.1	Simulink Layout	51
5.2.2	Case 1: Horizontal Circular Motion	52
5.2.3	Case 2: Vertical Circular Motion	55
5.2.4	Case 3: Straight Line Motion	57
5.3	Discussion	60
6	Connecting Path Segments	63
6.1	The Usability of PCTP	63
6.2	Simulations	64
6.2.1	Case 1: Horizontal Circle Connected to a Straight Line	65
6.2.2	Case 2: Vertical Circle Connected to a Straight Line	67
6.2.3	Case 3: Horizontal Circle Connected to a Vertical Circle	70
6.3	Discussion	74
7	Connections with Sharp Corners	75
7.1	Corners in Pick-and-Place Motions	75
7.2	Sharp Corners in Path Constrained Motion	76
7.2.1	Simulations with Transition Segment	80
7.2.2	Design Options	82
7.3	Discussion	86
8	Concluding Remarks	87
	Bibliography	89
A	Derivations and Definitions	91

A.1	Order Reduction for Matlab Simulations	92
A.2	Transformation Matrix	93
A.3	Jacobian for Linear Velocity	94
A.4	Inertia Matrices	94
A.5	SolidWorks Parameter Estimation	95
A.6	Velocity Profile Generation Points	95
B	Maple Code	97
C	MATLAB: Simulink	113
D	MATLAB: Level-2 S-function	117
E	ABB Datasheet	123

List of Figures

1.1 Industrial robotic production plant in Tianjin, China (Great Wall Motor Company). Image courtesy of the ABB Group.	2
1.2 General set of components in a robotic system.	3
1.3 CAD model of the ABB IRB140 robot manipulator in initial position. The CAD model can be downloaded from the ABB web site. [5]	4
3.1 Assignment of coordinate frames as basis for DH parameter representation of a simplified version of the 6DOF IRB140 manipulator.	18
3.2 Illustrating the geometrical view of the robot. The figure to the left is used for finding θ_1 , while the figure to the right is used for finding θ_2 and θ_3	20
4.1 Comparing 3 types of cubic splines, using the CurveFitting Toolbox in Matlab. The resulting interpolations show <i>csapi</i> (blue dashed line), <i>csape</i> (red dashed line) and <i>pchip</i> (black line) over the same set of points.	31
4.2 Velocity profile for an example path coordinate using the natural cubic spline, <i>csape</i> (red dashed line) closing in on the boundaries (blue lines).	32
4.3 Geometrical view of the robot in the xy -plane. The blue circle is centered on (x_{c1}, y_{c1}, z_{c1}) with a radius R_1 . $\theta_1 \in [-\pi, \pi]$	33
4.4 Geometrical view of the robot in the xz -plane. Blue line represents a circle in the xy -plane with radius R_1 , centered on (x_{c1}, y_{c1}, z_{c1})	34
4.5 Showing q_i^* as functions of $\theta_1 \in [-\pi, \pi]$	36
4.6 Velocity constraints along the horizontal circular path.	36
4.7 Comparing three types of cubic splines on four different velocity profile scenarios for the horizontal circular path. <i>csapi</i> (blue dashed), <i>csape</i> (red dashed), <i>pchip</i> (black).	37
4.8 Showing planned velocity profiles for the horizontal circular case, with various end point settings, together with the velocity constraints. Red dashed line shows the chosen Spline. Black, blue and green lines show boundary lines for q_1 , q_2 and q_3 respectively.	38
4.9 Geometrical view of the robot in the xz -plane. Blue line represents a circle in the yz -plane with radius R_2 , centered on (x_{c2}, y_{c2}, z_{c2})	40

4.10	Showing q_i^* as functions of $\theta_2 \in [-\pi, \pi]$ for a vertical circle with radius R_2 , centered on (x_{c2}, y_{c2}, z_{c2})	41
4.11	Boundary curves for the vertical circle path.	41
4.12	Comparing three types of cubic splines on four different velocity profile scenarios for the vertical circular path. <i>csapi</i> (blue dashed), <i>csape</i> (red dashed), <i>pchip</i> (black).	42
4.13	Showing planned velocity profiles for the vertical circular case, with various end point settings, together with the velocity constraints. Red dashed line shows the chosen Spline. Black, blue and green lines show boundary lines for q_1 , q_2 and q_3 respectively.	43
4.14	Geometrical view of the robot in the xz -plane. Blue line represents a straight line with length R_1	44
4.15	Showing q_i^* as functions of $x \in [0.45, 0.65]$, for the straight line path	45
4.16	Boundary curves for the straight line path case.	45
4.17	Comparing three types of cubic splines on four different velocity profile scenarios for the straight line path. <i>csapi</i> (blue dashed), <i>csape</i> (red dashed), <i>pchip</i> (black).	46
4.18	Showing planned velocity profiles for the straight line case, with various end point settings, together with the velocity constraints. Red dashed line shows the chosen Spline. Black, blue and green lines show boundary lines for q_1 , q_2 and q_3 respectively.	46
5.1	Simulink model of the IRB140 with orbital stabilization control and a motion planning structure.	51
5.2	Time evolution of θ_1 for the velocity profile where starting and ending velocities are set to zero.	52
5.3	Obtained velocity profile, $\dot{\theta}_1$, generated by cubic spline interpolation for the horizontal circle path.	53
5.4	Showing obtained trajectory for the horizontal circle in joint space. q_1 (black), q_2 (blue), q_3 (green). Dashed lines show the desired trajectory. <i>Left figure</i> : Starting and ending velocity set to zero. <i>Right figure</i> : Starting and ending velocity set to boundary level.	53
5.5	Showing obtained trajectory for the horizontal circle in task space. $x(t)$ (red), $y(t)$ (blue), $z(t)$ (black). Dashed lines show the desired trajectory. <i>Left figure</i> : Starting and ending velocity set to zero. <i>Right figure</i> : Starting and ending velocity set to boundary level.	54
5.6	Showing the error vector for the horizontal circle path, Y_{hc} , i.e the error between the desired $\Phi(\theta_1)$ and obtained $q(t)$. y_1 (red), y_2 (blue), y_3 (black). <i>Left figure</i> : Starting and ending velocity set to zero. <i>Right figure</i> : Starting and ending velocity set to boundary level.	54
5.7	Time evolution of $\theta_2 \in [-\pi, \pi]$	55

5.8	Resulting velocity profile for the vertical circle, obtained by cubic spline interpolation.	56
5.9	Showing the obtained (continuous) trajectories with the desired trajectories (dashed) from simulations for the vertical circular path. Upper figures show task space trajectories: $x(t)$ (red), $y(t)$ (blue), $z(t)$ (black). Lower figures show joint space trajectories: q_1 (black), q_2 (blue), q_3 (green). Figures to the left show the result of having boundary level velocities at the end points, while the figures to the right show the same result using zero end point velocities.	56
5.10	Showing the error vector for the vertical circle path, Y_{vc} , i.e the error between the desired $\Phi(\theta_2)$ and obtained $q(t)$. y_1 (red), y_2 (blue), y_3 (black). <i>Left figure</i> : Starting and ending velocity set to zero. <i>Right figure</i> : Starting and ending velocity set to boundary level.	57
5.11	Time evolution for the straight line path with the velocity profile set to zero starting and ending velocity.	57
5.12	Resulting velocity profile with zero starting and ending velocity for the straight line path, obtained by cubic spline interpolation.	58
5.13	Showing the obtained (continuous) trajectories with the desired trajectories (dashed) from simulations for the straight line path. Upper figures show task space trajectories: $x(t)$ (red), $y(t)$ (blue), $z(t)$ (black). Lower figures show joint space trajectories: q_1 (black), q_2 (blue), q_3 (green). Figures to the left show the result of having boundary level velocities at the end points, while the figures to the right show the same result using zero end point velocities.	59
5.14	Showing the error vector for the straight line path, Y , i.e the error between the desired $\Phi(x)$ and obtained $q(t)$. y_1 (red), y_2 (blue), y_3 (black). <i>Left figure</i> : Starting and ending velocity set to zero. <i>Right figure</i> : Starting and ending velocity set to boundary level.	59
6.1	Example of a path badly suited for path constrained trajectory planning, similar to the example seen in [10].	64
6.2	Resulting trajectory in joint space of combining a horizontal circle and a straight line path. The velocity profiles are set with low starting and ending velocities for both path segments. Dashed lines show desired trajectory.	65
6.3	Resulting trajectory in task space of combining a horizontal circle and a straight line path. The velocity profiles are set with low starting and ending velocities for both path segments. Dashed lines show desired trajectory.	66
6.4	Error between desired and obtained joint values on the combined horizontal circle and straight line path.	66

6.5	Resulting trajectory in joint space of combining a horizontal circle and a straight line path. The velocity profiles are set with boundary level velocities at the connection point for both path segments. Dashed lines show desired trajectory.	67
6.6	Resulting trajectory in joint space of combining a vertical circle and a straight line path. The velocity profiles are set with low starting and ending velocities for both path segments. Dashed lines show desired trajectory.	68
6.7	Resulting trajectory in task space of combining a vertical circle and a straight line path. The velocity profiles are set with low starting and ending velocities for both path segments. Dashed lines show desired trajectory.	68
6.8	Error between desired and obtained joint values on the combined vertical circle and straight line path.	69
6.9	Resulting trajectory in joint space of combining a vertical circle and a straight line path. The velocity profiles are set with boundary level velocities at the connection point for both path segments. Dashed lines show desired trajectory.	69
6.10	Resulting trajectory in joint space of combining a horizontal circle and a vertical circle path. <i>Upper figure:</i> Velocity profiles are set to zero at the connection point between the two paths. <i>Lower figure:</i> Velocity profiles are set to boundary level velocity through the connection point. Dashed lines show desired trajectory.	71
6.11	Resulting trajectory in task space of combining a horizontal circle and a vertical circle path. <i>Upper figure:</i> Velocity profiles are set to zero at the connection point between the two paths. <i>Lower figure:</i> Velocity profiles are set to boundary level velocity through the connection point. Dashed lines show desired trajectory.	72
6.12	Resulting error vector when combining a horizontal circle and a vertical circle path. <i>Upper figure:</i> Velocity profiles are set to zero at the connection point between the two paths. <i>Lower figure:</i> Velocity profiles are set to boundary level velocity through the connection point.	73
7.1	Illustration of the blending procedure developed by Lloyd and Hayward [9]. Figure from [23].	76
7.2	Combining two path segments using a third segment designed as a suitable transition path.	77
7.3	Rounding corner between two path segments for path constrained motion. The desired path is shown in blue. The new path will follow the red circle from the point where the dashed line meets the blue curved path segment and continue up the straight line from where the red circle touches the straight line path segment.	78

7.4	Illustrating the geometry of a circular segment. Illustration from Wikipedia.	79
7.5	Showing planned $q_i^*(\theta_t)$ for the transition segment with radius $r_t = 0.04$ (m). We see that the initial positions align with the corresponding positions of $q_i^*(\theta_1)$ for $\theta_1 = \beta$.	80
7.6	Resulting trajectory in joint space as functions of $\theta \in [-\pi, \beta]$ after implementing the transition path segment. Transition segment is using lower velocity profile, resulting in higher path accuracy. Highlighted area (between the red lines) shows the area of effect from the transition segment.	81
7.7	Resulting trajectory in task space after implementing the transition path segment.	81
7.8	Highlighting the effect of tuning the transition segment radius for the obtained $y(t)$ (blue) with desired trajectory (dashed blue). <i>Top left:</i> $r_t = 0.03$. <i>Top right:</i> $r_t = 0.035$. <i>Bottom left:</i> $r_t = 0.04$. <i>Bottom right:</i> $r_t = 0.08$. (m)	82
7.9	Error in task space after implementing the transition path segment, showing the result for four different r_t of the transition segment. <i>Top left:</i> $r_t = 0.03$. <i>Top right:</i> $r_t = 0.035$. <i>Bottom left:</i> $r_t = 0.04$. <i>Bottom right:</i> $r_t = 0.08$. (m)	83
7.10	Highlighting the effect of tuning the transition segment velocity profile on the obtained $y(t)$ (blue) with desired trajectory (dashed blue). <i>Top left:</i> zero velocity profile settings. <i>Top right:</i> Low velocity profile settings. <i>Bottom left:</i> velocity profile equal to horizontal circle velocity profile. <i>Bottom right:</i> higher velocity profile settings.	84
7.11	Error vector in task space after implementing the transition path segment, showing the result for four different velocity profile settings on the transition segment.	85
7.12	A more robust way of implementing, adding more possible transition paths with various level of precision. The circle with a thicker line shows the same circle as in Figure 7.3.	85
A.1	Showing how the mass center of link 3 was calculated in SolidWorks.	95
C.1	Simulink model - Overall structure of the simulation of orbital stabilization on the motion planning cases	114
C.2	IRB140 block - Containing a Level-2 MATLAB S-Function block for simulating the dynamics of the IRB140 manipulator.	114
C.3	Controller block - Taking in the desired and obtained joint values for orbital stabilization. Also containing a block for friction handling.	114
C.4	Trajectory block - Structure of the trajectory generation, is generated in the Velocity profile block, the desired trajectory signal is then computed in a Matlab function.	115

C.5	Example of trajectory generation block for the straight line - Velocity profile is generated in the Velocity profile block, the desired trajectory signal is then computed in a Matlab function.	115
C.6	Velocity profile block - Simple structure of the interpolation by the look-up-table block.	116
C.7	Motion generator block structure - Taking the vector $q(t)$ as input. . . .	116

List of Tables

3.1	DH parameter table	18
3.2	Known lengths from datasheet	19
3.3	Masses and centre of mass calculated by SolidWorks for the simplified 3DOF model of the IRB140.	23
4.1	Constraints for position (q_i) and velocity (\dot{q}_i)	30
4.2	Points used in the cubic spline interpolation in Figure 4.1.	31
5.1	Planned periods of motion, T , for various end point velocity profiles. $T_{f,i}$ shows the periods of motion for end point velocities set to boundary levels, $T_{s,i}$ shows the periods of motion for end point velocities set to zero, $T_{v,s}$ shows the periods of motion for zero initial velocity and boundary level final velocity, $T_{r,s}$ shows the periods of motion for boundary level initial velocity and zero final velocity.	61
5.2	Obtained periods of motion, T , for various end point velocity profiles. $T_{f,i}$ shows the periods of motion for end point velocities set to boundary levels, $T_{s,i}$ shows the periods of motion for end point velocities set to zero, $T_{v,s}$ shows the periods of motion for zero initial velocity and boundary level final velocity, $T_{r,s}$ shows the periods of motion for boundary level initial velocity and zero final velocity.	61
A.1	Points used in the cubic spline interpolation for the horizontal circle. . .	95
A.2	Points used in the cubic spline interpolation for the vertical circle. . . .	95
A.3	Points used in the cubic spline interpolation for the straight line with boundary level velocities at the starting and ending points.	96
A.4	Points used in the cubic spline interpolation for the straight line with zero velocities at the starting and ending points.	96
A.5	Points used in the cubic spline interpolation for the straight line with boundary level velocity at the starting point and zero at the ending point.	96

Chapter 1

Introduction

1.1 Motivation and Introduction to Industrial Robot Manipulators

A robot is a mechanical device, equipped with actuators and sensors, set to perform a certain task in an operating workspace. The ISO definition of an **industrial robot** is an automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes. [14] The ever growing need to make industry more efficient and safe has introduced robots as potential replacements for manual labour. More and more industries choose to have parts of their production work force consist of robots instead of humans. This year, the The International Federation of Robotics wrote a press release confirming the numbers of industrial robot sales (168 000 units) in 2013 and they predict an average increase of 6% per year starting between year 2014 and 2016. [1] Typical jobs for robots in the industry include welding, painting, product inspection, assembly and testing to name a few. As the technology advances, the requirements follow, and the desire to optimize the productivity is strong, since even small improvements can generate substantial profits in production. In Figure 1.1 we see a robotic production plant in Tianjin, China, where several manipulators are collaborating, doing various tasks on a car assembly line. These tasks should all be accomplished at high speed, with high endurance and precision.

In developing robotic systems, two major aspects can be distinguished, namely motion planning and motion control. In Figure 1.2 we see the general structure of a robotic system, where the motion planning components are highlighted. The purpose of robot motion planning is to generate a set of desired trajectories for each actuator of the system. This is often obtained by guiding the robotic system through a field of (dynamic) obstacles present in the workspace of the robot, while respecting the existing physical limitations. The desired trajectories serve as input to the motion controller. Mainly, the purpose of the motion controller is to achieve

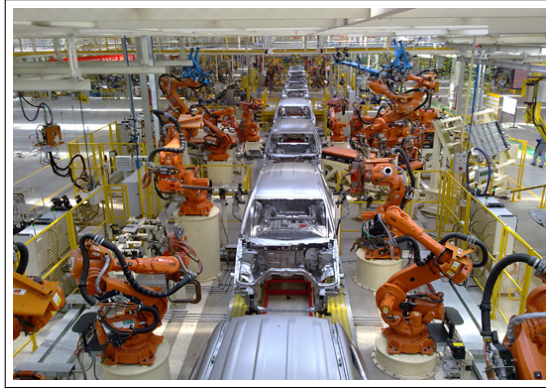


Figure 1.1: Industrial robotic production plant in Tianjin, China (Great Wall Motor Company). Image courtesy of the ABB Group.

accurate and robust tracking of the given desired trajectories. Over the last decades, a huge amount of work has been done in the field of robotic motion control. We refer to [7] for a relatively updated and complete overview.

In this text we will focus on methods for motion planning with respect to time optimization, as well as study a motion control strategy that achieves high precision tracking for preplanned trajectories.

1.1.1 The IRB140 Robot Manipulator

Throughout the text, we will be using the robot IRB140 as our working example robot. The IRB140 is an industrial robot produced by the ABB Group. The manipulator has six revolute joints, all controlled by AC-motors (6 degrees of freedom). A Computer-aided design (CAD) model of the robot can be seen in Figure 1.3, showing its default initial position. The manipulator is suited for welding, painting, assembly and other tasks that require high speed and accuracy. It also has the possibility to be mounted on walls and roofs, with its relatively modest size and around 100 kg of weight. The datasheet given by ABB, is included in the file folder related to this paper, and is also available on the web. A compressed version of the datasheet can be seen in Appendix E.

1.2 Scope and Emphasis

The main contribution of the thesis is a study of motion planning of near time optimal trajectories when combining path segments, which may be useful in many situations where robotic industrial production applies. We have planned velocity assignments based on velocity and geometric constraints, with the goal of obtaining time optimal

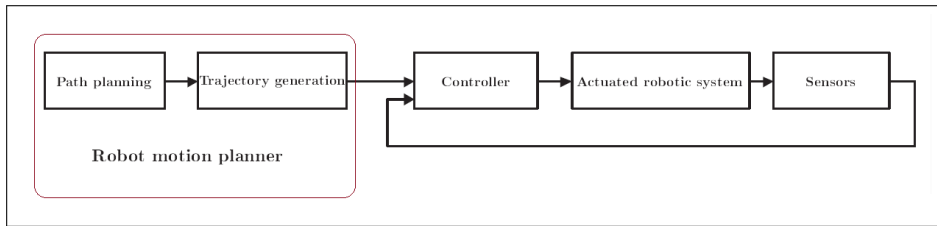


Figure 1.2: General set of components in a robotic system.

periods of motion along preplanned paths. In most of the literature found on the subject, the initial and final velocity is set to zero for any given path segment. We will investigate the possibilities of connecting two (or more) paths into new and longer paths without having to set the velocity to zero at the connection point between the path segments. During this study, a special case surfaced, where two path segments are connected in such a fashion that they generate sharp corners. Some time is given to finding a new way of dealing with such cases. To ensure high precision path tracking, we will study a state-of-the-art motion control strategy based on orbital stabilization. The control structure, as well as mathematical models of the manipulator, were implemented in Simulink, where we simulated the manipulator behavior on several case paths with various velocity assignment profiles. The mathematical modeling of the manipulator robot is presented as a compressed and corrected version of the project work leading up to this thesis. (See [13] for this project report.)

1.3 Outline of Report

The report is laid out in the following way: Chapter 2 deals with the theoretical background that is needed to follow the text, explaining various common methods and concepts that will be used later. In Chapter 3, some of the concepts from Chapter 2 will be used to derive mathematical models of our working example robot, IRB140. In Chapter 4, the problem of path and trajectory planning is concerned. We will introduce 3 case paths, and the problem of trajectory planning is handled thoroughly for each case, using path constrained trajectory planning. Then, in Chapter 5 we relate the motion planning results with dynamics when we deal with the motion control problem of the IRB140 on the paths. We will use a controller based on orbital stabilization, and perform simulations on the desired trajectories. In Chapter 6, we will analyze the possibilities of connecting the presented paths into new and longer paths. We will see how the periods of motion for the new trajectories compare to the results from the motion planning stage. Lastly, in Chapter 7 we make an attempt to solve some issues concerning sharp corners, discovered from the results of the analysis in Chapter 6.

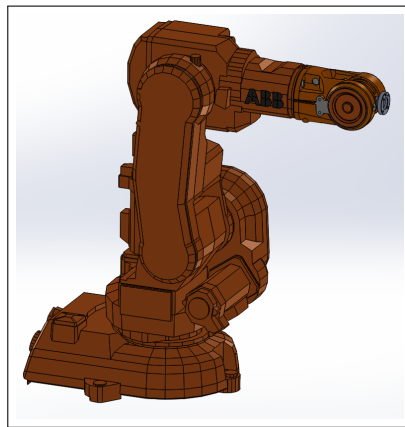


Figure 1.3: CAD model of the ABB IRB140 robot manipulator in initial position. The CAD model can be downloaded from the ABB web site. [5]

Chapter 2

Concepts and Theories

This chapter briefly explains the governing theories and methods used in this paper. We start in Section 2.1 by introducing some mathematical modeling methods related to the kinematics of robot manipulators. The mathematical modeling is continued in Section 2.2 where we focus on the dynamics of manipulators. In Section 2.3 we focus on path and trajectory planning, and clarify some common terminology. Then, in Section 2.4 we investigate some theory related to robot motion control.

2.1 Robot Manipulator Kinematics

In this section we introduce common methods for mathematically describing the kinematics of robot manipulators with **degree of freedom**, n (n -DOF). Note that many different notations can be found in the literature when dealing with mathematical robotics. The notation used in this chapter will be similar to that of [26].

2.1.1 The Denavit-Hartenburg Convention

It is obvious that dealing with robot manipulators with n degrees of freedom can become very complex as n grows. Therefore, it is natural to seek a structured way to express robot joints and frames. The **Denavit-Hartenburg** (DH) Convention is a common way to achieve this goal. One reason why the DH convention is popular is because of its systematic nature, making engineers able to simplify complex situations and agree on a universal language with which they can communicate. [26, p. 76] We seek to find a homogeneous transformation matrix, $A_i = A_i(q_i)$, that expresses the position and orientation of frame $o_i x_i y_i z_i$ with respect to frame $o_{i-1} x_{i-1} y_{i-1} z_{i-1}$. Each homogeneous transformation, A_i , can be represented as a product of four basic transformations, as seen in Equation (2.1).

$$\mathbf{A}_i(\mathbf{q}_i) = \mathbf{Rot}_{z,\theta_i} \mathbf{Trans}_{z,d_i} \mathbf{Trans}_{x,a_i} \mathbf{Rot}_{x,\alpha_i}, \quad i = 1 \dots n \quad (2.1)$$

The individual basic rotation matrices can be seen in [26] and any other book on robotics. Performing the calculations on Equation (2.1), gives the matrix as shown in Equation (2.2).

$$\mathbf{A}_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) \cos(\alpha_i) & \sin(\theta_i) \sin(\alpha_i) & a_i \cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \cos(\alpha_i) & -\cos(\theta_i) \sin(\alpha_i) & a_i \sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.2)$$

The matrix A_i , will then be of the form in (2.3), where the matrix R_i^{i-1} represents the rotation between the two frames $o_i x_i y_i z_i$ and $o_{i-1} x_{i-1} y_{i-1} z_{i-1}$. The vector \mathbf{o}_i^{i-1} represents the coordinate vector between the frames.

$$\mathbf{A}_i = \begin{bmatrix} \mathbf{R}_i^{i-1} & \mathbf{o}_i^{i-1} \\ 0 & 1 \end{bmatrix} \quad (2.3)$$

To obtain the matrix A_i it can be seen from (2.2) that we need to find the parameters a_i , α_i , d_i and θ_i . Here a_i is the **link length**, α_i is the **link twist**, d_i is the **link offset**, and θ_i is the **joint angle**. These parameters can be easily obtained if we arrange the frame definitions correctly according to the DH convention. In [26, p. 110], the following algorithm is presented for assigning frames

1. Establish a base frame. Set the origin anywhere on the z_0 -axis. The x_0 and y_0 axes are chosen conveniently to form a right-handed frame.
2. Locate the origin o_i where the common normal to z_i and z_{i-1} intersects z_i . If z_i intersects locate o_i at this intersection. If z_i and z_{i-1} are parallel, locate o_i in any convenient position along z_i .
3. Establish x_i along the common normal between z_{i-1} and z_i through o_i , or in the direction normal to the $z_{i-1} - z_i$ plane if z_{i-1} and z_i intersect.
4. Establish y_i to complete a right handed frame.
5. Establish the end effector frame $o_n x_n y_n z_n$.
6. Create a table of DH parameters a_i , d_i , α_i and θ_i .

By doing these steps, we can readily obtain values for the parameters. The link length will be the distance along x_i from the intersection of the x_i and z_{i-1} axes to o_i . The link twist is the angle from z_{i-1} to z_i measured about x_i . The link offset will be the distance along z_{i-1} from o_{i-1} to the intersection of the x_i and z_{i-1} axes. If the joint is prismatic, d_i will be a variable, which is denoted d_i^* . Lastly, the joint angle is the angle from x_{i-1} to x_i measured about z_{i-1} . If joint i is revolute, θ_i will be a variable, and denoted θ_i^* .

2.1.2 Forward Kinematics

Looking at (2.3) it is clear it would be more useful to find a way to represent any frame, most often the base frame, with respect to any other frame. This can be obtained simply by multiplication of the homogeneous matrices, A_i , in order, starting and ending with the desired frames. Performing this multiplication we can now express the position and orientation of $o_jx_jy_jz_j$ with respect to $o_ix_iy_iz_i$, by a **transformation matrix**, T_j^i . The most interesting transformation matrix is often the one who express the position and orientation of the end effector with respect to the base frame $o_0x_0y_0z_0$.

$$\mathbf{T}_n^0 = \mathbf{A}_1(\mathbf{q}_1) \dots \mathbf{A}_n(\mathbf{q}_n) \quad (2.4)$$

where \mathbf{T}_n^0 is of the form

$$\mathbf{T}_n^0 = \begin{bmatrix} \mathbf{R}_n^0 & \mathbf{o}_n^0 \\ 0 & 1 \end{bmatrix} \quad (2.5)$$

This matrix together with the DH convention defines the forward kinematic equations for a manipulator and makes us able to map from joint variables to end effector position and orientation. However, to be able to control a manipulator, the inverse problem needs to be solved, i.e. given position and orientation of the end effector, we must solve for the corresponding joint variables.

2.1.3 Inverse Kinematics

The problem of inverse kinematics can be stated as follows: [26, pp. 93-94] Find a solution, or possibly multiple solutions, of the equation

$$T_n^0(q_1, \dots, q_n) = H \quad (2.6)$$

where H is a given 4x4 homogeneous transformation $\in SE(3)$, and $T_n^0(q_1, \dots, q_n) = A_1(q_1) \dots A_n(q_n)$.

The problem of inverse kinematics is in general more difficult than the forward kinematics problem. This is not hard to understand, considering the complexity of the nonlinear trigonometric equations that needs to be solved with high n . However, there are techniques to simplify the problem, like geometric approaches and kinematic decoupling methods. In this paper the geometric approach will be used in several different situations for the ABB IRB140 manipulator.

2.2 Robot Manipulator Dynamics

The previous sections describe the motion of robots without consideration of the torques and forces that produce the motion. In this section we deal with the dynamics of robot manipulators, which explicitly describe the relationship between motion and force. This relation is essential in being able to implement control algorithms on a robot manipulator. What follows is the derivation presented in [26, pp. 240-241], of a set of differential equations, namely the **Euler-Lagrange Equations**, making us able to understand the dynamics of the robot manipulator.

2.2.1 The Euler Lagrange Equations

To arrive at the Euler-Lagrange Equations, we begin by inspecting Newton's second law

$$\frac{d(mv)}{dt} = f \quad (2.7)$$

For a particle with mass, m , constrained to move vertically, with downward gravity force, mg , (2.7) becomes

$$m\ddot{y} = f - mg \quad (2.8)$$

where the left hand side of (2.8) can be written as

$$m\ddot{y} = \frac{d}{dt}(m\dot{y}) = \frac{d}{dt} \frac{\partial}{\partial \dot{y}} \left(\frac{1}{2}m\dot{y}^2 \right) = \frac{d}{dt} \frac{\partial \mathcal{K}}{\partial \dot{y}} \quad (2.9)$$

Here, $\mathcal{K} = \frac{1}{2}m\dot{y}^2$ is the kinetic energy. Similarly, the gravitational force in (2.8) can be expressed as

$$mg = \frac{\partial}{\partial y}(mgy) = \frac{\partial \mathcal{P}}{\partial y} \quad (2.10)$$

where $\mathcal{P} = mgy$, is the potential energy due to gravity. We now define a **Lagrangian**, $\mathcal{L} = \mathcal{K} - \mathcal{P}$, and observe that if we differentiate the Lagrangian with respect to y and \dot{y} , we obtain

$$\frac{\partial \mathcal{L}}{\partial y} = -\frac{\partial \mathcal{P}}{\partial y} \quad (2.11)$$

and

$$\frac{\partial \mathcal{L}}{\partial \dot{y}} = \frac{\partial \mathcal{K}}{\partial \dot{y}} \quad (2.12)$$

We are now able to rewrite (2.8) as

$$\frac{d}{dt} \frac{\partial \mathcal{L}}{\partial \dot{y}} - \frac{\partial \mathcal{L}}{\partial y} = f \quad (2.13)$$

Equation (2.13) is called the **Euler-Lagrange Equation** for the example of the particle in the xy -plane, moving parallel to the y -axis. A robot manipulator with n degrees of freedom, can be described by a similar set of equations, using so called **generalized coordinates** (q_1, \dots, q_n) as follows:

$$\frac{d}{dt} \frac{\partial \mathcal{L}}{\partial \dot{q}_k} - \frac{\partial \mathcal{L}}{\partial q_k} = \tau_k, \quad k = 1, \dots, n \quad (2.14)$$

where τ_k is the force associated with q_k . By using (2.14), we can now describe the dynamics of the robot manipulator. The question of deriving these equations for the robot still remains, and will be clarified in Section 2.2.3. Let us first rewrite (2.14) to a convenient form which is often used in the literature of robotics. In [26, pp. 255-257] it is shown how to arrive at the following form of (2.14)

$$\sum_{i=1}^n m_{ij}(q) \ddot{q}_j + \sum_{j=1}^n \sum_{k=1}^n c_{ijk}(q) \dot{q}_j \dot{q}_k + g_i(q) = \tau_i \quad (2.15)$$

Here, the terms c_{ijk} are known as the Christoffel symbols, which are found by

$$c_{ijk} = \frac{1}{2} \left\{ \frac{\partial d_{kj}}{\partial q_i} + \frac{\partial d_{ki}}{\partial q_j} - \frac{\partial d_{ij}}{\partial q_k} \right\} \quad (2.16)$$

The form in (2.15) is useful due to its linearity in the $\dot{q}_j \dot{q}_k$ terms, making for a convenient way of dealing with trajectory generation. Another form of (2.14), that is often used, is by matrix representation

$$M(q)_{n \times n} \begin{bmatrix} \ddot{q}_1 \\ \vdots \\ \ddot{q}_n \end{bmatrix} + C(q, \dot{q})_{n \times n} \begin{bmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_n \end{bmatrix} + G(q)_{n \times 1} = \begin{bmatrix} \tau_1 \\ \vdots \\ \tau_n \end{bmatrix}. \quad (2.17)$$

where the matrix $M(q)$ is called the **Inertia Matrix**, and is symmetric and positive definite for any manipulator, making it useful for control problems. The matrix $C(q, \dot{q})$ is the **Coriolis-Centrifugal Matrix** and $G(q)$ is known as the **Gravity Vector**.

2.2.2 Holonomic Constraints

Before we look at methods for deriving the dynamic equations, let us introduce some terminology concerning constraint analysis regarding (2.14). If the particle from the example in Section 2.2.1, joined by a new set of particles, is free to move about without any restrictions, then it is easy to describe their motion by Newton's second law. However, if the motions of the particles are constrained in some fashion, then one must take into account not only the externally applied forces, but also the constraint forces. With this in mind, we can now introduce the term of **holonomic constraints**. [26, p. 244] A constraint on the k coordinates, r_1, \dots, r_k is called holonomic if it is an equality constraint of the form

$$g_i(r_1, \dots, r_k) = 0, \quad i = 1, \dots, l. \quad (2.18)$$

By differentiating (2.18) we get an expression of the form

$$\sum_{j=1}^k \frac{\partial g_i}{\partial r_j} \cdot dr_j = 0 \quad (2.19)$$

The holonomic constraints will be important in Chapter 4 when we look at path constrained trajectory planning, using **virtual** holonomic constraints.

2.2.3 The Recursive Newton-Euler Formulation

In this section we will see how to derive the dynamic equations from Section 2.2.1 for robotic manipulators. There are mainly two methods for obtaining this goal, namely the Lagrangian formulation and the Newton-Euler formulation. As the title suggest, we are going to focus on the Newton-Euler formulation in this paper. It should be pointed out that both the formulations are equivalent in almost all respects, although they build upon two different philosophies. While the Lagrangian method treats the manipulator as a whole and perform the analysis using the Lagrangian function, the Newton-Euler formulation treat each link of the robot in turn, writing down the equations for linear and angular motion. In [26, p. 272] it is argued that the Lagrangian formulation might be more suitable if one is interested in obtaining closed-form equations that describe the time evolution of the generalized coordinates. The Newton-Euler formulation is suggested to be more suited if we are interested in knowing what generalized forces need to be applied in order to realize a *particular* time evolution of the generalized coordinates. For this paper, however, it is not of much difference which formulation is chosen.

The Newton-Euler formulation is explained in detail in [26, pp. 271-279], and only the most important parts are restated here. It should also be noted that several errors are found in this section of [26], so some care should be taken while using the

2006 version of this book. First, we introduce a set of vectors and parameters which are listed below.

$a_{c,i}$	acceleration of the center of mass of link i
$a_{e,i}$	acceleration of the end of link i
ω_i	angular velocity of frame i with respect to frame 0
α_i	angular acceleration of frame i with respect to frame 0
g_i	acceleration due to gravity
f_i	the force exerted by link $i-1$ on link i
τ_i	torque exerted by link $i-1$ on link i
R_{i+1}^i	rotation matrix from frame i to frame $i+1$
z_i	axis of actuation of frame i with respect to frame 0
m_i	the mass of link i
I_i	inertia tensor of link i about a frame parallel to frame i
$r_{i-1,ci}$	vector from origin of frame $i-1$ to the center of mass of link i
$r_{i,ci}$	vector from the origin of frame i to the center of mass of link i
$r_{i-1,i}$	vector from the origin of frame $i-1$ to the origin of frame i

The force balance for a single link i rises the following equation based on Newton's second law

$$f_i - R_{i+1}^i f_{i+1} + m_i g_i = m_i a_{c,i} \quad (2.20)$$

Next, one can compute the moment balance equation for link i . The moment exerted by a force f about a point is given by $f \times r$, where r is the radial vector from the point where the force is applied to the point about which we are computing the moment. In the moment equation, the vector $m_i g_i$ does not appear, since it is applied directly at the center of mass. Thus, we have

$$\tau_i - R_{i+1}^i \tau_{i+1} + f_i \times r_{i-1,ci} - (R_{i+1}^i f_{i+1}) = \omega_i \times (I_i \omega_i) + I_i \alpha_i \quad (2.21)$$

The main point in the Newton-Euler formulation consists of finding vectors f_1, \dots, f_n and τ_1, \dots, τ_n corresponding to a given set of vectors q, \dot{q}, \ddot{q} . In other words, we find the forces and torques in the manipulator that correspond to a given set of generalized coordinates and their first two derivatives. The general idea is as follows. Given q, \dot{q}, \ddot{q} , suppose we can determine all of the velocities and accelerations of various parts of the manipulator, that is, all of the quantities $a_{c,i}, \omega_i$ and α_i . Then we can solve (2.20) and (2.21) recursively to find all the forces and torques as follows. First, set $f_{n+1} = 0$ and $\tau_{n+1} = 0$. This expresses the fact that there is no link $n+1$. We can then solve (2.20) and (2.21). Repeating this for each link in order, will then

give a full solution once we find an easily computed relation between q, \dot{q}, \ddot{q} and $a_{c,i}, \omega_i$, and α_i . This can be obtained by a recursive procedure in the direction of increasing i . For the angular velocity we get

$$\omega_i^{(0)} = \omega_{i-1}^{(0)} + z_{i-1} \dot{q}_i \quad (2.22)$$

where the superscript (0) denotes that ω_i is express in the inertial frame. We can now use the rotation of joint i expressed in frame i , $b_i = (R_i^0)^T R_{i-1}^0 z_0$, to get

$$\omega_i = (R_i^{i-1})^T \omega_{i-1} + b_i \dot{q}_i \quad (2.23)$$

Next, for the angular acceleration, the time derivate of (2.22) gives

$$\dot{\omega}_i^{(0)} = \dot{\omega}_{i-1}^{(0)} + z_{i-1} \ddot{q}_i + \omega_i^{(0)} \times z_{i-1} \dot{q}_i \quad (2.24)$$

which can be expressed in frame i as

$$\alpha_i = (R_i^{i-1})^T \alpha_{i-1} + b_i \ddot{q}_i + \omega_i \times b_i \dot{q}_i \quad (2.25)$$

The next step is to find an expression for $a_{e,i}$ and $a_{c,i}$. The linear velocity of the mass center of link i is

$$v_{c,i}^{(0)} = v_{e,i-1}^0 + \omega_i^{(0)} \times r_{i-1,ci}^{(0)} \quad (2.26)$$

which gives

$$a_{c,i}^{(0)} = a_{e,i-1}^{(0)} \times r_{i-1,ci}^{(0)} + \omega_i^{(0)} \times (\omega_i^{(0)} \times r_{i-1,ci}^{(0)}) \quad (2.27)$$

With this, we are ready to express the acceleration of the center of mass of link i in frame i as follows

$$a_{c,i} = (R_i^{i-1})^T a_{e,i-1} + \dot{\omega}_i \times r_{i-1,ci} + \omega_i \times (\omega_i \times r_{i-1,ci}) \quad (2.28)$$

where we have used the fact that the rotation matrix is skew symmetric, giving the property $R(a \times b) = Ra \times Rb$. Finding $a_{e,i}$ can now be done simply by substituting the vector $r_{i-1,i}$ for $r_{i-1,ci}$ and arrive at the expression

$$a_{e,i} = (R_i^{i-1})^T a_{e,i-1} + \dot{\omega}_i \times r_{i-1,i} + \omega_i \times (\omega_i \times r_{i-1,i}) \quad (2.29)$$

These equations concludes the recursive Newton-Euler formulation, which is now done in two steps. Consider a robot manipulator with n links. We begin with setting the initial conditions of $\omega_0, \alpha_0, a_{c,0}$ and $a_{e,0}$ equal to zero. Also, the terminal conditions of f_{n+1} and τ_{n+1} should be set equal to zero. Next, we perform the **forward recursion**, solving the equations (2.23), (2.25), (2.29) and (2.28) in the given order for each link from 1, to n . Finally, we perform the **backward recursion** by solving (2.20) and (2.21) for each link, starting from link n , and ending with link 1. Completing this recursion, we will have obtained all the needed expressions to form the dynamic equations in (2.15).

2.3 Motion Planning Terminology

In this section we address the planning of a motion for a robot manipulator, and clarify some related terminology used throughout the paper. The detailed methods are best explained using a case robot with obtained kinematic relations, and will be saved for Chapter 4. When considering robotic systems set to perform specific tasks, we can distinguish between two types of movements. Movements can in some cases be specified in which only a certain set of points within the robot workspace must be visited. These types of movements represent what is called **pick-and-place** motion. In other cases movements can be set to follow a certain path exactly. This is called **path-constrained** motion.

The pick-and-place motion represents a movement of the robot where two desired positions are given. Firstly, the position where an object, e.g. a factory product, is picked. Secondly, the position where the object is to be placed. The user of such a motion planner provides the pick and place positions, as well as intermediate positions to avoid collisions during the motion. There are numerous algorithms found in literature to address this type of trajectory generation problem, and algorithms for such motions will often be implemented for tasks such as e.g. component mounting and packing tasks, where there exists freedom in the design of the path between the pick and place positions.

In path constrained motions, a path is typically presented as a function of a path parameter, meaning the motion can be described in one degree-of-freedom. [17] Implementing such motion generation will often be used in tasks such as e.g. cutting and inspection. The path parameter can be used to describe the process and actuator constraints by a set of functions. In this paper we will be focusing mainly on path-constrained motion for preplanned paths where the paths will be made up of common geometrical shapes such as circular motions and straight lines. These paths will then be used to study the problem of time optimization.

In literature, several terms related to motion planning are used. Consider the motion planning for a robot manipulator, starting at an initial position and ending at a goal position. Given specified initial and end configurations, **path planning** is the problem of finding a *collision free path* connecting these configurations. By its description, this problem may sound like an easy task, yet the path planning problem is among the most difficult problems in computer science. [26, p. 163] The computational complexity of the best known complete path planning algorithm grows exponentially with the number of internal degrees of freedom of the robot. As the purpose of this paper is to investigate methods for time optimal control, the path planning problem will be reduced to simple feasible predefined paths.

Trajectory planning is a term that has been used for decades in robotics, and refers mainly to the problem of determining both a path and a velocity function for a robot arm. [10, p. 792] As one can expect, the velocity profile on a path will come in under concern when addressing time optimality. Methods for finding a suitable velocity profile that minimizes time, is something that has been investigated since the 1980s. One such method uses constraint-based analysis to reach a sub-optimal solution to the problem, and is called **path-constrained trajectory planning**. This method will be investigated thoroughly throughout the paper. Path-constrained trajectory planning is also useful outside the scope of traditional industry robots. In [15], the method is used for a machine crane, with a discussion regarding the reverse motion. For further discussion and other methods see e.g [2] and [6].

In the discussion of trajectory planning, we will need to know some basic interpolation techniques, in particular the **cubic spline** will be used in this paper. A spline is defined as a sufficiently smooth piecewise polynomial interpolant. Sufficiently smooth means, in this case, that we need to mandate the order of the spline to fit our needs for a given curve fitting problem. A spline function $g(x)$ of order p with knots t_1, \dots, t_J , is presented in [3, p. 247] on the form

$$g(x) = \sum_{j=1}^{J+p+1} \beta_j G_j(x) \quad (2.30)$$

where

$$\begin{cases} G_j(x) = (x - t_j)_+^p, & j = 1, \dots, J \\ G_{J+j}(x) = x^{j-1} & j = 1, \dots, p+1 \end{cases}$$

A *cubic* spline will be such a function, of order, $p = 3$. Splines on this form are frequently used in practice. There are several different types of cubic splines with different end point settings, which will be discussed further in Section 4.2.3.

Lastly, let us clarify some terminology related to the mathematical spaces that will be used in this paper. The transformations explained in Section 2.1.2 make us able to remap our mathematical models between spaces. The 3 dimensional Euclidean space, denoted \mathcal{W} , will be called the world or task space. A robot in this space is denoted \mathcal{A} (or $\mathcal{A}_1, \dots, \mathcal{A}_n$ for linkage). An obstacle region is denoted \mathcal{O} and a configuration space is denoted by \mathcal{C} . A smooth manifold \mathcal{X} , called the *state space*, may be $\mathcal{X} = \mathcal{C}$ or it may be a phase space derived from \mathcal{C} if dynamics are considered. This notation is similar to what is found in commonly used literature, e.g. [10].

2.4 Motion Control Theory

Let us revisit the result from Section 2.2 where we found the differential equations in matrix form, (2.17), for the robot dynamics.

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = u \quad (2.31)$$

where u is the input torque. A basic **PD controller** can be applied, where

$$u = -K_p(q^* - q) - K_d(\dot{q}^* - \dot{q}) = -K_p e - K_d \dot{e} \quad (2.32)$$

$$e = (q^* - q), \quad \dot{e} = (\dot{q}^* - \dot{q}) \quad (2.33)$$

where q^* denotes the desired trajectory. It can be shown that this control strategy yields asymptotic tracking, by using a Lyapunov function candidate on the form

$$V = \frac{1}{2}\dot{q}^T M(q)\dot{q} + \frac{1}{2}e^T K_p e \quad (2.34)$$

thus, for a constant reference, i.e. $\dot{q}^* = 0$, we have a Lyapunov function candidate that satisfies $V > 0 \quad \forall (q, \dot{q}) \neq (q^*, 0)$, at which $V = 0$. Next we can show that $\dot{V} \leq 0 \quad \forall \dot{q} \neq 0$, which implies the system is heading towards equilibrium.

$$\dot{V} = \dot{q}^T M(q)\ddot{q} + \frac{1}{2}\dot{q}^T \dot{M}(q)\dot{q} + \dot{q}^T K_p e \quad (2.35)$$

$$\dot{V} = \dot{q}^T (u - C(q, \dot{q})\dot{q} - G(q)) + \frac{1}{2}\dot{q}^T \dot{M}(q)\dot{q} + \dot{q}^T K_p e \quad (2.36)$$

$$\dot{V} = \dot{q}^T (u - G(q) + K_p e) + \frac{1}{2}\dot{q}^T (\dot{M}(q) - 2C(q, \dot{q}))\dot{q} \quad (2.37)$$

$$(2.38)$$

Using the skew symmetric properties of $\dot{M}(q) - 2C(q, \dot{q})$, as can be seen in [26, p. 121], we see that the term $\dot{q}^T (\dot{M}(q) - 2C(q, \dot{q}))\dot{q}$ disappears. We can then insert the controller equations to get

$$\dot{V} = \dot{q}^T (u - G(q) + K_p e) \quad (2.39)$$

$$\dot{V} = -\dot{q}^T K_d \dot{q} \leq 0 \quad \forall \dot{q} \neq 0 \quad (2.40)$$

Next, suppose $\dot{V} \equiv 0$, i.e. \dot{V} is equal to 0 for all instants. Then $\dot{q} \equiv 0$, which implies $\ddot{q} \equiv 0$ if K_d is chosen to be positive definite. This means $e = 0$, and what follows from La Salle's theorem (see e.g. [25]) is proof of global asymptotic stability for $q = q^*$. This result will be used in Section 5.1 where we will expand on these results and introduce orbital asymptotic stability.

Chapter 3

Mathematical Modeling of the IRB140 Manipulator

In this chapter we consider mathematical modeling of the IRB140 presented in Section 1.1.1. Based on the methods and concepts from Chapter 2, we will derive the kinematics and dynamics of the robot manipulator. Some of the results presented here will be similar to those obtained in the project work leading up to this thesis. [13]

3.1 Kinematics

As stated in Section 2.1, **Forward kinematics** makes us able to describe the position and orientation of an end effector given values for the joint variables of the robot. **Inverse kinematics** is focused on determining the values of the joint variables when knowing the end effector position and orientation. Lastly, **velocity kinematics**, makes us able to write down a Jacobian for the system.

3.1.1 Forward Kinematics

The IRB140 has 6 degrees of freedom ($n=6$). The manipulator has convenient placement of the last 3 joints, making it able to reach every point in its workspace, by only using its first 3 degrees of freedom, i.e. the first 3 joints. By exploiting this fact, we can introduce a simplification of the robot from $n=6$ to $n=3$, reducing the degrees of freedom to 3. The resulting simplified model is illustrated in Figure 3.1. Following Section 2.1.1, the coordinate frames were assigned according to the rules of the DH convention, as seen in Figure 3.1.

Note that the initial positions of the joints are chosen such that link 2 is vertical and link 3 is horizontal. This means that the x_3 axis is locked in horizontal direction at the end effector. The resulting DH parameter table from the analysis can be found in Table 3.1, where the numerical values of the lengths a_i and d_i can be found from the datasheets supplied by ABB [4], which is included in Appendix E.

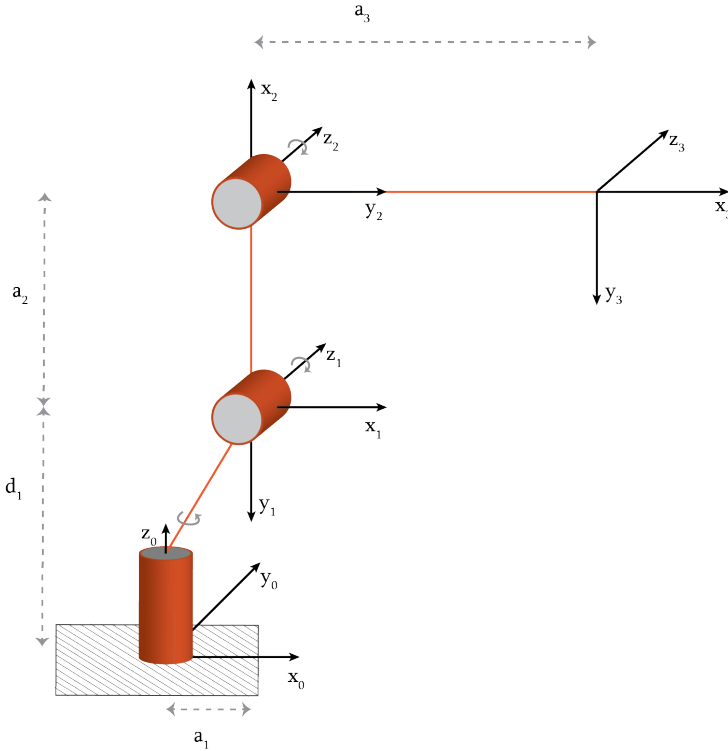


Figure 3.1: Assignment of coordinate frames as basis for DH parameter representation of a simplified version of the 6DOF IRB140 manipulator.

Having found the parameters, each homogeneous transformation, A_i , can now be represented as a product of four basic transformations as explained in Section 2.1.1. First, let us note that the θ_i -parameters contain difference and sum angles for θ_2 and θ_3 respectively. Since the calculations later on will become very complex, these expressions were compressed using common trigonometric identities before stating the A_i -matrices. The 3 homogeneous transformation matrices for the IRB140 are

Table 3.1: DH parameter table

Link	a_i	α_i	d_i	θ_i
1	a_1	$-\frac{\pi}{2}$	d_1	q_1^*
2	a_2	0	0	$q_2^* - \frac{\pi}{2}$
3	a_3	0	0	$q_3^* + \frac{\pi}{2}$

found to be

$$\mathbf{A}_1 = \begin{bmatrix} \cos(q_1) & 0 & -\sin(q_1) & a_1 \cos(q_1) \\ \sin(q_1) & 0 & \cos(q_1) & a_1 \sin(q_1) \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

$$\mathbf{A}_2 = \begin{bmatrix} \sin(q_2) & \cos(q_2) & 0 & a_2 \sin(q_2) \\ -\cos(q_2) & \sin(q_2) & 0 & -a_2 \cos(q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.2)$$

$$\mathbf{A}_3 = \begin{bmatrix} -\sin(q_3) & -\cos(q_3) & 0 & -a_3 \sin(q_3) \\ \cos(q_3) & -\sin(q_3) & 0 & a_3 \cos(q_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.3)$$

Now, having defined A_1 , A_2 and A_3 , we can use (2.4) to obtain the transformation matrix that relates the end effector frame to the base frame. The calculation of \mathbf{T}_3^0 is done using Maple, and can be seen in Appendix B. The resulting transformation matrix takes the form seen in (3.4). The individual matrix elements, $r_{p,q}$, for $p, q = 1, 2, 3$, as well as d_x , d_y and d_z of (3.4) can be seen in (A.1). Having obtained the link lengths from the datasheet, which are stated in Table 3.2, we see that the distances from the origin to the point (d_x, d_y, d_z) of the end effector can now be described by the joint angles q_i .

$$\mathbf{T}_3^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & d_x \\ r_{21} & r_{22} & r_{23} & d_y \\ r_{31} & r_{32} & r_{33} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.4)$$

Table 3.2: Known lengths from datasheet

d_1	0.352 m
a_1	0.070 m
a_2	0.360 m
a_3	0.445 m

3.1.2 Inverse Kinematics

In this section we solve the problem of inverse kinematics which is stated in (2.6), for the IRB140. We will be looking at a geometric approach to the general inverse solution to this problem for the manipulator. In later chapters this geometrical approach will be used on predefined paths.

The geometric approach starts by analyzing the projections into the plane, as can be seen from Figure 3.2. The inverse kinematic solution can be then be analytically derived. θ_1 is straight forward to write down from the figure. The two-argument arctangent function is used for easier computer programming capability. For θ_1 and θ_3 , we also have to consider the 'elbow-up' and 'elbow-down' solutions. The base joint, $q_1 \in [-\pi, \pi]$, is mechanically constrained (See the datasheet [4]), so there will not be multiple solutions other than the two 'elbow-up' and 'elbow-down' solutions for q_2 and q_3 .

$$\theta_1 = A \tan 2(\pm d_y, \pm d_x) \quad (3.5)$$

θ_3 is found by using the law of cosines, while also considering θ_1 (see Figure 3.2), to obtain

$$\cos \theta_3 = \frac{s^2 + r^2 - a_2^2 - a_3^2}{2a_2a_3} \quad (3.6)$$

which becomes

$$\cos \theta_3 = \frac{(d_x + a_1 \cos(\theta_1))^2 + (d_y + a_1 \sin(\theta_1))^2 + (d_z - d_1)^2 - a_2^2 - a_3^2}{2a_2a_3} := D \quad (3.7)$$

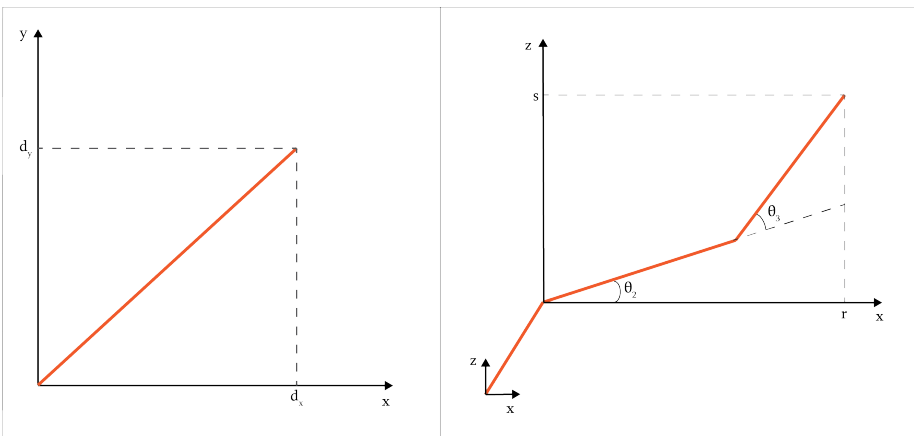


Figure 3.2: Illustrating the geometrical view of the robot. The figure to the left is used for finding θ_1 , while the figure to the right is used for finding θ_2 and θ_3 .

Taking care of the 'elbow-up' and 'elbow-down'-solutions we get

$$\theta_3 = A \tan 2(D, \pm \sqrt{1 - D^2}) \quad (3.8)$$

And lastly we find θ_2 as

$$\begin{aligned} \theta_2 = & A \tan 2(d_z - d_1, \sqrt{(d_x + a_1 \cos(\theta_1))^2 + (d_y + a_1 \sin(\theta_1))^2}) \\ & - A \tan 2(a_3 \sin(\theta_3), a_2 + a_3 \cos(\theta_3)) \end{aligned} \quad (3.9)$$

It is now possible to determine q_1, q_2 and q_3 by investigating our defined initial position, i.e. the position as can be seen in Figure 3.1. Notice that the angle equations in this case contain the nonlinear trigonometric equations from from \mathbf{T}_3^0 , which can be seen in (A.1), making for a relatively complex problem even for a manipulator with 3 degrees of freedom.

3.1.3 Velocity Kinematics

In this section we derive the velocity relationships of the IRB140, relating the linear and angular velocities of the end effector to the joint velocities. Mathematically, the forward kinematic equations define a function between the space of Cartesian positions and orientations and the space of joint positions. The velocity relationships are then determined by the **Jacobian** of this function. The Jacobian is one of the most important quantities in the analysis and control of robot motion. [26]. For finding a coordinate vector for the cartesian position of the flange with respect to the base frame, we can use

$$\mathbf{P}^0 = \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_{3 \times 1} \end{bmatrix} \mathbf{T}_3^0 \begin{bmatrix} \mathbf{0}_{3 \times 1} \\ 1 \end{bmatrix} \quad (3.10)$$

which gives the vector containing the position elements of \mathbf{T}_0^3 as

$$\mathbf{P}^0 = \begin{bmatrix} -\cos(q_1)[a_3 \sin(q_2) \sin(q_3) - a_3 \cos(q_2) \cos(q_3) - a_2 \sin(q_2) - a_1] \\ -\sin(q_1)[a_3 \sin(q_2) \sin(q_3) - a_3 \cos(q_2) \cos(q_3) - a_2 \sin(q_2) - a_1] \\ -a_3 \cos(q_2) \sin(q_3) - a_3 \sin(q_2) \cos(q_3) + a_2 \cos(q_2) + d_1 \end{bmatrix} \quad (3.11)$$

Now it is possible to find the Jacobian for linear velocity. This is done by calculating the partial derivate of \mathbf{P}^0 , to get the linear velocity of the end effector. Now using the fact that the Jacobian for linear velocity must satisfy

$$\mathbf{v}_n^0 = \mathbf{J}_v \dot{\mathbf{q}} \quad (3.12)$$

we obtain the Jacobian for linear velocity, expressed as a 3×3 -matrix

$$\mathbf{J}_{\mathbf{v}3 \times 3} = \begin{bmatrix} \dot{j}_{11} & \dot{j}_{12} & \dot{j}_{13} \\ \dot{j}_{21} & \dot{j}_{22} & \dot{j}_{23} \\ \dot{j}_{31} & \dot{j}_{32} & \dot{j}_{33} \end{bmatrix} \quad (3.13)$$

The individual matrix elements of (3.13) can be seen in Appendix A.3. Similarly, the Jacobian for angular velocity could be found using $\omega_n^0 = J_\omega \dot{q}$, to obtain the full Jacobian matrix $J = [J_v, J_\omega]^T$.

3.2 Dynamics

In the previous sections we analysed the kinematics of the IRB 140. The kinematic equations describe the robot motion without consideration of the forces and torques producing the motion. To be able to control the manipulator to behave in a desired manner, both concepts needs to be accounted for. The goal of this section is to find the **dynamic equations** for the manipulator (IRB140), which explicitly describes the relationship between force and motion.

3.2.1 Parameter Estimation

Section 2.2 explained how the dynamic model can be found by a recursive algorithm known as the Newton-Euler formulation. It was described how the algorithm requires computation of a set of equations related to velocity and acceleration. To be able to use set up the formulation, we first need to find some estimates for the different components used in the equations. As we can see in (2.23), (2.25), (2.29) and (2.28), we need to find some vectors that describe the frame positions relative to the mass centers of the links. Also, we need to find matrices for the inertia tensor.

3.2.2 Finding Vectors and Mass Centers

Unfortunately, the producer company of the IRB140, ABB, does not include all the needed information in the datasheet. However, on their website ([5]) it is possible to download CAD-models for the robot, which can be used to calculate the masses and mass centers using commercial software, such as SolidWorks. This was done using the Mass Properties function in SolidWorks, where we have the possibility to combine links to account for our simplification from a 6-DOF to a 3-DOF model. This can be seen in Section A.5. The results from this analysis can be seen in Table 3.3. The datasheet from ABB claims the total mass of the robot to be 98kg. Calculations in SolidWorks show the total mass of the links as ≈ 71.5 kg, this leaves 26.5kg to the base, which seems reasonable.

Table 3.3: Masses and centre of mass calculated by SolidWorks for the simplified 3DOF model of the IRB140.

Link	Mass(in kg)	Centre of Mass (in mm)		
1	34.655	$x = 89.03$	$y = -27.87$	$z = 43.12$
2	15.994	$x = 198.29$	$y = -92.43$	$z = 9.73$
3	20.862	$x = 79.96$	$y = 4.56$	$z = 5.86$

Now, having found the mass centers, it is possible to determine the vectors that was introduced in Section 2.2.3. Note that the mass centers stated here are based on the distance from frame $i - 1$ to the center of mass of link i .

Vectors from the origin of frame $i-1$ to the origin of frame i :

$$\vec{r}_{0,1} = \begin{bmatrix} a_1 & -d_1 & 0 \end{bmatrix}^T$$

$$\vec{r}_{1,2} = \begin{bmatrix} a_2 & 0 & 0 \end{bmatrix}^T$$

$$\vec{r}_{2,3} = \begin{bmatrix} a_3 & 0 & 0 \end{bmatrix}^T$$

Vectors from the origin of frame $i-1$ to the center of mass of link i :

$$\vec{r}_{0,c1} = \begin{bmatrix} 0.08903 & -0.02789 & 0.04312 \end{bmatrix}^T$$

$$\vec{r}_{1,c2} = \begin{bmatrix} 0.19829 & -0.09243 & 0.00973 \end{bmatrix}^T$$

$$\vec{r}_{2,c3} = \begin{bmatrix} 0.07996 & 0.00456 & 0.00586 \end{bmatrix}^T$$

Vectors from the origin of frame i to the center of mass of link i :

$$\vec{r}_{1,c1} = \vec{r}_{0,c1} - \vec{r}_{0,1}$$

$$\vec{r}_{2,c2} = \vec{r}_{1,c2} - \vec{r}_{1,2}$$

$$\vec{r}_{3,c3} = \vec{r}_{2,c3} - \vec{r}_{2,3}$$

3.2.3 The Inertia Tensor

The inertia matrix often appears when dealing with angular momentum, kinetic energy and resultant torque of rigid systems. These are also used directly in the Newton-Euler algorithm, which is the goal of this section, so estimations of these are needed. SolidWorks gives the possibility to show the Inertia matrix for the links from the CAD-models, and these values have been used in this paper. The results show that the biggest values are found on the diagonals of the matrices, as expected.

This means the mass distribution is close to symmetric with respect to the frame. Another way of finding an estimate of the inertia matrices, without the use of CAD software, would therefore be to approximate the links to common shapes such as cylinders or cuboids and used the known inertia tensor expressions for such shapes.

3.2.4 The Newton-Euler Formulation

Having found the needed components, we can now derive the recursive Newton-Euler formulation for the IRB140 manipulator. Note that the expressions that come out of this recursion can be very large, even for a 3-DOF manipulator, and because of that, the resulting equations are not shown in this paper. However, the Maple code written to compute the equations, can be found in Appendix B, and is also included in the file folder related to this paper. Instead we will present the symbolic recursion in this section, showing how the recursion is derived for the IRB 140 with the found vectors, rotation matrices and parameters. The rotation matrices are calculated using the upper left 3×3 -matrices in A_i from (3.1)-(3.3). We begin by defining the initial gravity vector, g_0 to be

$$\vec{g}_0 = \begin{bmatrix} 0 & 0 & -g \end{bmatrix}^T$$

where $g = 9.81 \frac{m}{s^2}$, and the rotation axis vector z_0 ,

$$\vec{z}_0 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$$

according to our defined frames in Figure 3.1.

Now we can find the gravity vector related to each link by using

$$g_i = (R_i^0)^T g_0 \quad (3.14)$$

as well as the axis of rotation of joint i expressed in frame i by using

$$b_i = (R_i^0)^T z_{i-1} \quad (3.15)$$

where

$$z_i = R_i^0 z_0 \quad (3.16)$$

The vectors are calculated as seen in Appendix B.

Forward Recursion

We now continue with the forward recursion for the 3-DOF model of the IRB 140, starting with link 1 and ending with link 3, as follows

Link 1:

$$\begin{aligned}
\text{Angular velocity:} & \quad \omega_1 = b_1 \dot{q}_1 \\
\text{Angular acceleration:} & \quad \alpha_1 = b_1 \ddot{q}_1 + \omega_1 \times b_1 \dot{q}_1 \\
\text{Center acceleration:} & \quad a_{c,1} = \dot{\omega}_1 \times r_{0,c1} + \omega_1 \times (\omega_1 \times r_{0,c1}) \\
\text{End acceleration:} & \quad a_{e,1} = \dot{\omega}_1 \times r_{0,1} + \omega_1 \times (\omega_1 \times r_{0,1})
\end{aligned}$$

Link 2

$$\begin{aligned}
\text{Angular velocity:} & \quad \omega_2 = (R_2^1)^T \omega_1 + b_2 \dot{q}_2 \\
\text{Angular acceleration:} & \quad \alpha_2 = (R_2^1)^T \alpha_1 + b_2 \ddot{q}_2 + \omega_2 \times b_2 \dot{q}_2 \\
\text{Center acceleration:} & \quad a_{c,2} = (R_2^1)^T a_{e,1} + \dot{\omega}_2 \times r_{1,c2} + \omega_2 \times (\omega_2 \times r_{1,c2}) \\
\text{End acceleration:} & \quad a_{e,2} = (R_2^1)^T a_{e,1} + \dot{\omega}_2 \times r_{1,2} + \omega_2 \times (\omega_2 \times r_{1,2})
\end{aligned}$$

Link 3

$$\begin{aligned}
\text{Angular velocity:} & \quad \omega_3 = (R_3^2)^T \omega_2 + b_3 \dot{q}_3 \\
\text{Angular acceleration:} & \quad \alpha_3 = (R_3^2)^T \alpha_2 + b_3 \ddot{q}_3 + \omega_3 \times b_3 \dot{q}_3 \\
\text{Center acceleration:} & \quad a_{c,3} = (R_3^2)^T a_{e,2} + \dot{\omega}_3 \times r_{2,c3} + \omega_3 \times (\omega_3 \times r_{2,c3}) \\
\text{End acceleration:} & \quad a_{e,3} = (R_3^2)^T a_{e,2} + \dot{\omega}_3 \times r_{2,3} + \omega_3 \times (\omega_3 \times r_{2,3})
\end{aligned}$$

Backward recursion

Having found the expressions for the velocities and accelerations, we begin the backward recursion, starting from link 3, down to link 1, as follows

Link 3

$$\begin{aligned}
\text{Force on link:} & \quad f_3 = m_3 a_{c,3} - m_3 g_3 \\
\text{Torque:} & \quad \tau_3 = -f_3 \times r_{2,c3} + I_3 \alpha_3 + \omega_3 \times (I_3 \omega_3)
\end{aligned}$$

Link 2

$$\begin{aligned}
\text{Force on link:} & \quad f_2 = R_2^1 f_3 + m_2 a_{c,2} - m_2 g_2 \\
\text{Torque:} & \quad \tau_2 = R_2^1 \tau_3 - f_2 \times r_{1,c2} + R_2^1 f_3 \times r_{2,c2} + I_2 \alpha_2 + \omega_2 \times (I_2 \omega_2)
\end{aligned}$$

Link 1

$$\text{Force on link: } f_3 = m_3 a_{c,3} - m_3 g_3$$

$$\text{Torque: } \tau_1 = R_2^1 \tau_2 - f_1 \times r_{0,c1} + R_2^1 f_2 \times r_{1,c1} + I_1 \alpha_1 + \omega_1 \times (I_1 \omega_1)$$

3.3 Discussion

This concludes the mathematical modeling of the IRB140. In Section 3.1 we expressed the kinematics using common technique found in literature. In Section 3.2 we derived equations describing the dynamics of the manipulator using the Recursive Newton-Euler formulation. This resulted in a set of differential equations for the torque of each link of the simplified 3-link model of the IRB140. The resulting numerical equations from the recursions are very long and can be seen in length using the Maple code attached to this paper. In the project work leading up to this thesis, a similar model was validated by performing a simulation with a constant reference trajectory vector. This was done by implementing the model to a simple Simulink model in Matlab, where the reference vector was set to $q_d = [\frac{\pi}{6}, \frac{\pi}{4}, -\frac{\pi}{6}]^T$, and the initial vector was set to $q_0 = [0 \ 0 \ 0]^T$. We used a classic PD controller as presented in Section 2.4. The gain and damping matrices was set to $K_p = \text{diag}(150, 150, 150)$ and $K_d = \text{diag}(50, 50, 50)$ respectively. The updated and corrected model was validated in a similar manner for this paper, verifying that the model behaves as expected. See [13] to see the full procedure of how this was performed. It should be mentioned that for control purposes, we also need models for friction components.

Chapter 4

Motion Planning

The last chapter dealt with the mathematical modeling of the kinematics and dynamics of the IRB140 manipulator. The goal of this chapter is to investigate path constrained motion planning of the manipulator. We will study the problem of optimizing the time used to complete motions along various feasible paths. For this we will use a method called path-constrained trajectory planning, which is a method that avoids using the system dynamics in the planning process. We begin by looking at the underlying theories and concepts used by this method. Then we will use these concepts to perform motion planning on 3 different path cases. Also, we will investigate the use of spline interpolation for finding velocity assignments that produces (sub-)optimal velocity profiles subject along the paths. A huge amount of tools exists for generating splines in Matlab and Maple, and we will see the differences between those most relevant to the motion planning problem.

4.1 Path Planning

In robotics, a predefined path is a description of how the states of the robot should evolve. As mentioned in Section 2.3, the problem of path planning is among the most difficult problems in computer science. The main concern in this paper is time optimization, so the path planning problem can be simplified into finding a feasible predefined path that is subject to differential constraints. As mentioned in the introduction, a total of 3 such predefined path cases will be investigated in this chapter.

One often wish to find a parameterization of a path in task space and rewrite it in joint space. The definition of such path is a vector $\Phi(s)$, where $s = s(t)$ is a real and piecewise twice differentiable function, that works as a path parameter. The path has a starting point, $s(t_0) = s_0$ and an ending point $s(t_e) = s_e$. It is often

useful to parameterize the path as

$$\mathbf{P}^0 = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (4.1)$$

where \mathbf{P}^0 corresponds to the vector found for the end effector position in (3.11) by the expression

$$\mathbf{P}^0 = \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_{3 \times 1} \end{bmatrix} \mathbf{T}_3^0 \begin{bmatrix} \mathbf{0}_{3 \times 1} \\ 1 \end{bmatrix} \quad (4.2)$$

By using this parameterization, we can define a path coordinate, $s = s(t)$, as a function of the generalized coordinates that describe that particular path.

4.2 Path-Constrained Trajectory Planning

Having found a path coordinate, it is possible to define the desired path in the joint space. Given some path coordinate, $s = s(t)$, a path in the configuration space, \mathcal{C} , can be parameterized by

$$\Phi(s) = \begin{bmatrix} \phi_1(s) & \phi_2(s) & \cdots & \phi_n(s) \end{bmatrix}^\top. \quad (4.3)$$

Then it follows that the velocity and acceleration in joint space can be expressed as $\dot{\Phi}(s) = \Phi'(s)\dot{s}$ and $\ddot{\Phi}(s) = \Phi''(s)\dot{s}^2 + \Phi'(s)\ddot{s}$ respectively. When we assign a value, $s = s^*(t)$, such that the velocity along the trajectory behaves as desired, this becomes what is referred to as a motion generator, where the explicit dependence on time disappears. Taking this approach is called path-constrained trajectory planning, and a reason why this method is useful when our goal is time optimization will be highlighted in Section 4.2.1. See [10, pp. 846-856] for a deeper discussion on this topic. Let us for clarity write this parameterization mathematically as follows

$$q = \begin{bmatrix} q_1^*(t) \\ q_2^*(t) \\ \vdots \\ q_n^*(t) \end{bmatrix} = \Phi(s)|_{s=s^*(t)} = \begin{bmatrix} \phi_1(s) \\ \phi_2(s) \\ \vdots \\ \phi_n(s) \end{bmatrix} \Big|_{s=s^*(t)} \quad (4.4)$$

The variable, s , is now to be interpreted as a motion generator [10], which directly assigns the joint velocities and accelerations by

$$\dot{q} = \Phi'(s)\dot{s} \quad (4.5)$$

$$\ddot{q} = \Phi''(s)\dot{s}^2 + \Phi'(s)\ddot{s} \quad (4.6)$$

4.2.1 Time Optimization

Consider now the optimization problem that minimizes the traversal time along a path, starting at t_0 and ending at t_f , i.e

$$\underset{\tau}{\text{minimize}} \int_{t_0}^{t_f} dt \quad (4.7)$$

Solving such problems can become very difficult, especially when the workspace of robots have collision objects of different shapes and sizes. In general we want to perform optimization problems on convex sets rather than non-convex. The definition of a convex set in Euclidean space, \mathcal{W} , is that for every pair of points within an object, every point on a straight line segment between the points are also within the object. [12] It is clear that searching for a solution over the full path space does not fit this definition. However, there are methods for handling such challenges. The parameterization from the previous sections can be exploited to reformulate the optimization problem such that we get a change in the cost function. Using the chain rule we observe that

$$\frac{df}{dt} = \frac{df}{ds} \frac{ds}{dt} \implies dt = \frac{1}{\dot{s}} ds, \quad (4.8)$$

from which we see that minimizing the traversal time is equivalent to minimizing the inverse of the path coordinate velocity. This gives us a new formulation of the optimization problem

$$\underset{\tau(t)}{\text{minimize}} \int_{t_0}^{t_f} dt = \underset{s(t)}{\text{minimize}} \int_{s_0}^{s_f} \frac{1}{\dot{s}} ds, \quad (4.9)$$

where \dot{s} is a given velocity assignment along the trajectory.

With such a parameterization, the time evolution of the joints along the path for the robot manipulator, can be accessed by integrating a differential equation on the form

$$\dot{s}^{-1} = \frac{1}{h(s)} \quad (4.10)$$

The observation in (4.9) is very important, because together with (4.10), we can now close in on the optimal time solution by analyzing the boundary constraints.

It is worth noting that the cost function for the minimum time problem could be expanded to also consider power and energy consumption, by adding

$$\gamma_1 \int_{t_0}^{t_f} \left(\sum_{i=1}^n (\tau_i(t))^2 \right) dt \quad (4.11)$$

for energy, or

$$\gamma_2 \int_{t_0}^{t_f} \left(\sum_{i=1}^n (\dot{q}_i(t) \tau_i(t)) \right)^2 dt \quad (4.12)$$

for power consumption. These can be implemented to the cost function together with (4.7) by introducing the weight parameters γ_1 and γ_2 . However, the purpose of this paper is to only optimize with respect to time, i.e. $\gamma_1 = \gamma_2 = 0$.

4.2.2 Virtual Holonomic Constraints

In Section 2.2.2, we introduced the term holonomic constraints. As an example, the motion of a particle constrained to lie on a surface, for example a sphere, is subject to a holonomic constraint. If the particle is able to fall off the sphere under the influence of gravity, the constraint becomes non-holonomic. We have seen that equality constraints are of the form $g_i(r) = 0$, $i=1, \dots, l$. Now consider (4.4) where we synchronized the joints along the path to an independent configuration variable. If the geometric function in (4.4) is preserved by some control action along solutions of the closed-loop system, it is called a virtual holonomic constraint [20] [19] [18]. These constraints express relations among the generalized coordinates q_1, q_2 and q_3 for our case of the robot manipulator IRB140. For a feasible motion, such relations can always be found. [19, p. 2] If the system is fully actuated, the dynamics along the orbit of the motion is controlled, whereas for underactuated systems, it is not the case, the dynamics are fixed.

4.2.3 Finding Velocity Profiles

Naturally, robot manipulators will have velocity and acceleration constraints due to the limits on the actuators and the obtainable torque. The configuration is also usually constrained, and Table 4.1 shows the position and velocity constraints for the IRB140, as presented by the product manual from its producer ABB. These velocity constraint values are used in the rest of this paper. Note that it is straight forward to also account for the acceleration constraints linked to the system dynamics. For our case, however, these are found less restrictive than the velocity constraints, and are not to be accounted for here.

Table 4.1: Constraints for position (q_i) and velocity (\dot{q}_i)

Link	$q_{i,min}$ (rad)	$q_{i,max}$ (rad)	$\dot{q}_{i,min}$ (rad/s)	$\dot{q}_{i,max}$ (rad/s)
1	-3.1415	3.1415	-3.4907	3.4907
2	-1.5708	1.9199	-3.4907	3.4907
3	-4.0143	0.8727	-4.5379	4.5379

Table 4.2: Points used in the cubic spline interpolation in Figure 4.1.

Point	1	2	3	4	5	6	7	8	9	10	11
θ	$-\pi$	-2.4	-1.8	-1	-0.35	0	0.35	1	1.8	2.4	π
$\dot{\theta}$	1	1.5	5.6	6.5	8	8.5	8	6.5	5.6	1.5	1

In Section 4.2.1, we showed how minimizing traversal time is equivalent to minimizing the inverse of the path coordinate velocity. With this result, we can create the upper boundaries for the velocity profile by inspecting the boundary curves generated by the velocity constraints. Let \bar{q}_i be the maximum speed of joint i of the IRB140 manipulator. The boundary problem for the speed of each joint can then be expressed as

$$\|\dot{\phi}_i(s)\| \leq \|\phi'_i(s)\dot{s}\| \leq \bar{q}_i \quad (4.13)$$

Thus, the maximum value of \dot{s} can be found as

$$\max \|\dot{s}\| = \min_i \frac{\bar{q}_i}{\|\phi'_i(s)\|} \quad (4.14)$$

Now, the question of finding a suitable velocity profile still remains. The velocity profile should be suitable to fit (4.14). As explained earlier, a natural candidate would be a differential equation that relate s to \dot{s} . This make for effective constraint handling of each joint velocity. The differential equation as described would be of the form

$$\dot{s} = h(s) \quad (4.15)$$

Matlab has the possibility to implement cubic splines in Simulink, which would be suitable as candidates for the function $h(s)$. There are numerous choices that could

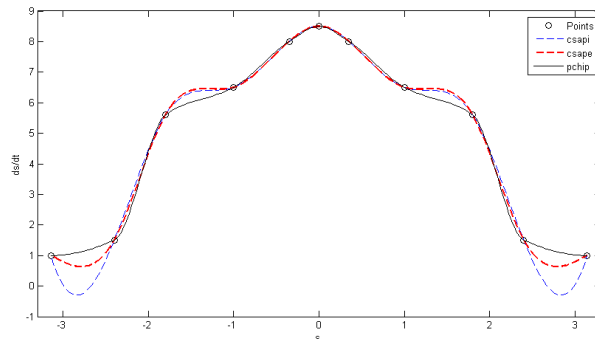


Figure 4.1: Comparing 3 types of cubic splines, using the CurveFitting Toolbox in Matlab. The resulting interpolations show csapi (blue dashed line), csape (red dashed line) and pchip (black line) over the same set of points.

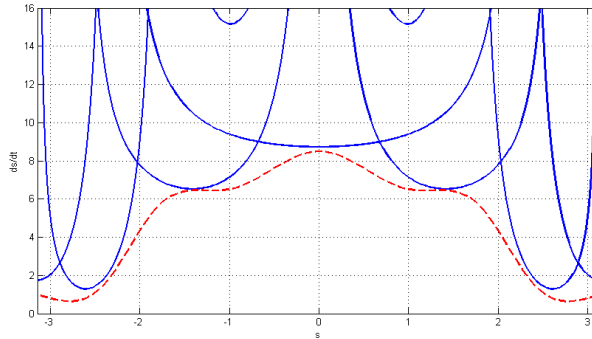


Figure 4.2: Velocity profile for an example path coordinate using the natural cubic spline, *csape* (red dashed line) closing in on the boundaries (blue lines).

give equally good results, such as Bezier polynomials. In Figure 4.1 3 different types of cubic splines are plotted on the same set of points. The black line shows the *pchip* function which can be found using the CurveFitting Toolbox in Matlab. This is a so-called Piecewise Cubic Hermite Interpolating Polynomial. Both the blue dashed line, which is produced using the *csapi* function, and the red dashed line, which is produced using the *csape* function produces cubic spline interpolants. The difference is found, as one can see on the figure, in the end point conditions. The *csapi* uses so-called not-a-knot end conditions, while the *csape* uses Lagrange end conditions.

We see that the *csapi* overshoots both the *pchip* and the *csape* at the end points. The *pchip* undershoots both *csapi* and *csape* both near the end points as well as the turning point near $s = 1$ and $s = -1$. It could be argued that the *pchip* is too flat for this situation. However, with rearranging and adding of more control points, it would probably give good results as well. The choice between *csapi* and *csape* seems arbitrary, given point modifications. With the *csape* one also has the possibility to choose other end conditions. The function *Spline* in Maple corresponds to the *csape* with Lagrange end conditions in Matlab. Figure 4.1 also show the points used when generating the splines, and the points are also listed in Table 4.2. The spline method used in this paper was chosen to be the *csape*, i.e. the cubic spline with Lagrange end points. The reason for this choice is that the Look-Up Table block in Simulink uses the cubic spline produced by the *csape* function, which makes for a handy way to implement the velocity profile in Simulink for simulations. The resulting spline, when using the points in Table 4.2 is plotted together with a set of velocity constraint curves in Figure 4.2, illustrating how this method can be used to close in on the velocity boundary constraints to generate near time optimal velocity profiles. In the following sections we will look at 3 different path cases and analyze them by applying the method of path constrained trajectory planning.

4.3 Motion Planning on Case Paths

Here we will present 3 case paths which will be subject to motion planning using the results and observations obtained from the previous sections. These path cases will be used throughout the rest of the thesis.

4.3.1 Case 1: Horizontal Circular Motion

In Section 3.1.2 we could see how the problem of inverse kinematics can become complex very quickly, even for a 3-DOF manipulator. The objective of this paper is to study time optimal control, so let us simplify the problem and introduce a case where the workspace is restricted to only one elbow-configuration. Consider a circle in the xy -plane with center at $(x_{c1} = 0.65, y_{c1} = 0, z_{c1} = 0.3)$ (m) and radius $R_1 = 0.2$ (m). Deriving the inverse kinematics of the end effector on such circle can be done analytically. Figure 4.3 illustrates the robot position in the xy -plane. The path can be parameterized as

$$\mathbf{P}_1^0 = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} R_1 \cos(\theta_1) + x_{c1} \\ R_1 \sin(\theta_1) \\ z_{c1} \end{bmatrix} \quad (4.16)$$

where \mathbf{P}_1^0 corresponds to the vector found for the end effector position in (3.11), by the expression

$$\mathbf{P}^0 = \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_{3 \times 1} \end{bmatrix} \mathbf{T}_3^0 \begin{bmatrix} \mathbf{0}_{3 \times 1} \\ 1 \end{bmatrix} \quad (4.17)$$

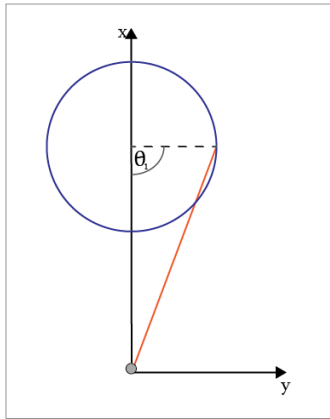


Figure 4.3: Geometrical view of the robot in the xy -plane. The blue circle is centered on (x_{c1}, y_{c1}, z_{c1}) with a radius R_1 . $\theta_1 \in [-\pi, \pi]$.

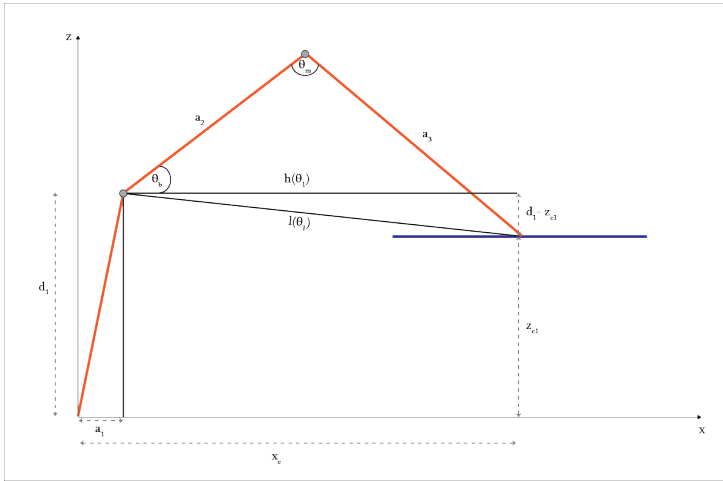


Figure 4.4: Geometrical view of the robot in the xz -plane. Blue line represents a circle in the xy -plane with radius R_1 , centered on (x_{c1}, y_{c1}, z_{c1}) .

Now, let us define a path coordinate as a function of the generalized coordinates that describe that path, e.g. the angular position

$$\theta_1 = \text{Atan2}(y(t), x(t) - x_{c1}) \quad (4.18)$$

From the figure it is easy to see that the x - and y -coordinates vary with θ_1 , where $\theta_1 \in [-\pi, \pi]$ is the angle defining the location along the circle. From this we can represent the first joint with respect to the angle θ_1 as follows (see ¹)

$$q_1^*(\theta_1) = \arctan(R_1 \sin(\theta_1)) , R_1 \cos(\theta_1) + x_{c1} \quad (4.19)$$

Now similar representations for the two last joints can be found. Figure 4.4 illustrates the robot position in the xz -plane. Let us define some of the lengths and angles in the figure using well known mathematical laws. First, the line $l(\theta_1)$ can be represented using the law of pythagoras

$$l(\theta_1) = \sqrt{(d_1 - z_{c1})^2 + (x_{c1} - a_1 + R_1 \cos(\theta_1))^2} \quad (4.20)$$

Secondly, the angle θ_m can be represented using the law of cosines

$$\theta_m = \arccos\left(\frac{a_3^2 + a_2^2 - l^2(\theta_1)}{2a_3a_2}\right) \quad (4.21)$$

Now, using the law of sines, the angle between a_2 and $l(\theta_1)$ can be found, making us able to represent θ_b as

$$\theta_b = \frac{a_3 \sin(\theta_m)}{l(\theta_1)} - \theta_f \quad (4.22)$$

¹The * in (4.19) denotes a desired joint angle value

where θ_f is the angle between the lines $h(\theta_1)$ and $l(\theta_1)$, and can be expressed as

$$\theta_f = \arctan\left(\frac{d_1 - z_{c1}}{x_{c1} - a_1 + R_1 \cos(\theta_1)}\right) \quad (4.23)$$

Finally, q_2 and q_3 can, along with q_1 , be expressed in terms of a single variable, θ_1 . Using the defined initial positions as seen in Figure 3.1, we get

$$q_2^*(\theta_1) = \frac{\pi}{2} - \theta_b \quad (4.24)$$

$$q_3^*(\theta_1) = \frac{\pi}{2} - \theta_m \quad (4.25)$$

Observing the solution to the inverse kinematics problem in this case, we see that the nonlinear equations from the forward kinematics analysis are avoided. It is obvious that restricting the end effector to follow a simple predefined geometric figure such as a circle gives big benefits in terms of simplicity, and more importantly it makes for a convenient parameterization of the joints.

As mentioned, the joint angles are functions of a single variable, θ_1 , i.e. $q_i^* = q_i^*(\theta_1)$. Taking the time dependency of θ_1 along the desired path, i.e. $\theta_1 = \theta_1^*(t)$, we see that we have obtained a parameterization of the path that fits the form of (4.3), where $\theta_1^*(t)$ is a real and twice differentiable function of time. Defining the functions that parameterize each joint $q = \Phi(\theta_1)$, we complete the parameterization, meaning (4.4) will take the following form for the IRB140 on the horizontal circle:

$$q = \begin{bmatrix} q_1^*(t) \\ q_2^*(t) \\ q_3^*(t) \end{bmatrix} = \Phi(\theta_1)|_{\theta_1=\theta_1^*(t)} = \begin{bmatrix} \phi_1(\theta_1) \\ \phi_2(\theta_1) \\ \phi_3(\theta_1) \end{bmatrix} \Big|_{\theta_1=\theta_1^*(t)} \quad (4.26)$$

Figure 4.5 shows the resulting evolution for the target trajectory as functions of the path parameter, and θ_1 is now to be interpreted as a motion generator (see e.g. [10]), which directly assign the joint velocities and accelerations by

$$\dot{q} = \Phi'(\theta_1)\dot{\theta}_1 \quad (4.27)$$

$$\ddot{q} = \Phi''(\theta_1)\dot{\theta}_1^2 + \Phi'(\theta_1)\ddot{\theta}_1 \quad (4.28)$$

For the horizontal circle path, the minimization problem can then be described simply by substituting the path coordinate from (4.9) and (4.10) with θ_1 to obtain

$$\underset{\theta_1(t)}{\text{minimize}} \int_{\theta_{1,0}}^{\theta_{1,f}} \frac{1}{\theta_1} d\theta_1 \quad (4.29)$$

where

$$\dot{\theta}_1 = h(\theta_1) \quad (4.30)$$

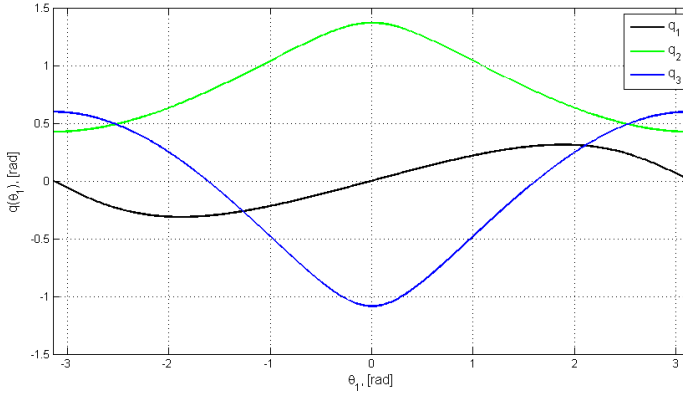


Figure 4.5: Showing q_i^* as functions of $\theta_1 \in [-\pi, \pi]$.

Now we can state the boundary problem expression from (4.13) as

$$\|\dot{\phi}_i(\theta_1)\| \leq \|\phi'_i(\theta_1)\dot{\theta}_1\| \leq \bar{q}_i \quad (4.31)$$

and the maximum value of $\dot{\theta}$ can be found as

$$\max \|\dot{\theta}_1\| = \min_i \frac{\bar{q}_i}{\|\phi'_i(\theta_1)\|} \quad (4.32)$$

The resulting boundary curve plot was made in Matlab and can be seen in Figure 4.6. Note that acceleration constraints are not concerned, due to them being

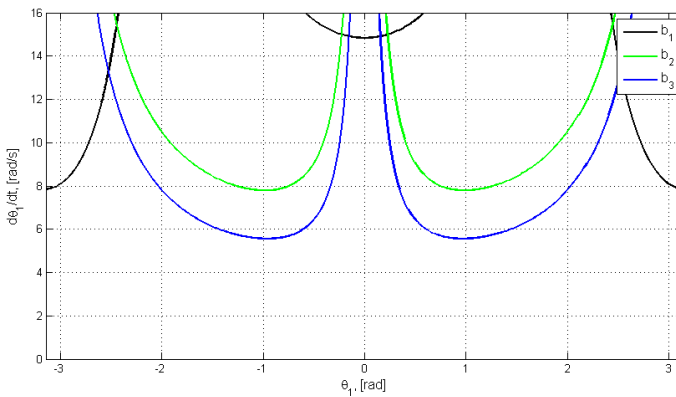


Figure 4.6: Velocity constraints along the horizontal circular path.

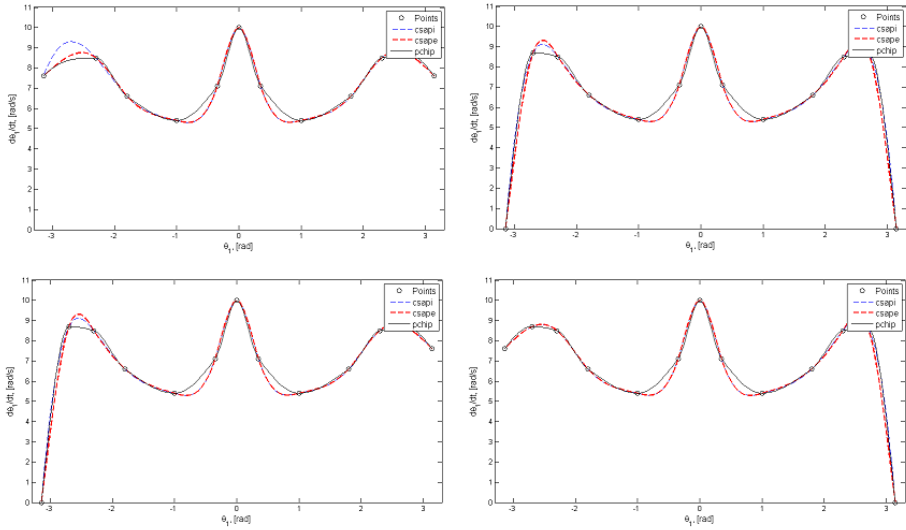


Figure 4.7: Comparing three types of cubic splines on four different velocity profile scenarios for the horizontal circular path. *csapi* (blue dashed), *csape* (red dashed), *pchip* (black).

less restrictive than the velocity constraints, as explained in [18]. The calculations of $\Phi(\theta_1)$ were first made in Maple, and then implemented to Matlab, and the code for the calculations can be seen in Appendix B. We see that in this case, the boundary curve, b_2 (green line), corresponding to the constraint on the second joint is less restrictive than b_1 (black line) and b_3 (blue line) throughout the complete motion from $\theta_1 = -\pi$ to $\theta_1 = \pi$. At the end points of the motion, b_1 works as the velocity constraint boundary, as well as around $\theta = 0$. Between these points, b_3 works as the constraint boundary.

The spline functions generated by Matlab can be seen in Figure 4.7 for four different velocity profiles. The upper left figure shows the planning result when using near time optimal velocity at the end points. The upper right figure shows the planning result of having zero velocity at the end points. The bottom figures show the results of planning with one end point set to zero velocity and the other set to near time optimal velocity. We can see that the planned velocity profiles are symmetrical around $\theta_1 = 0$ for the first two cases, while for the last two cases the velocity profiles are asymmetrical. It can be argued from the results in the figure that both the *csape* and the *csapi* splines can be used as our spline generator for this case, whereas the *pchip* seems to be too flat near the end points.

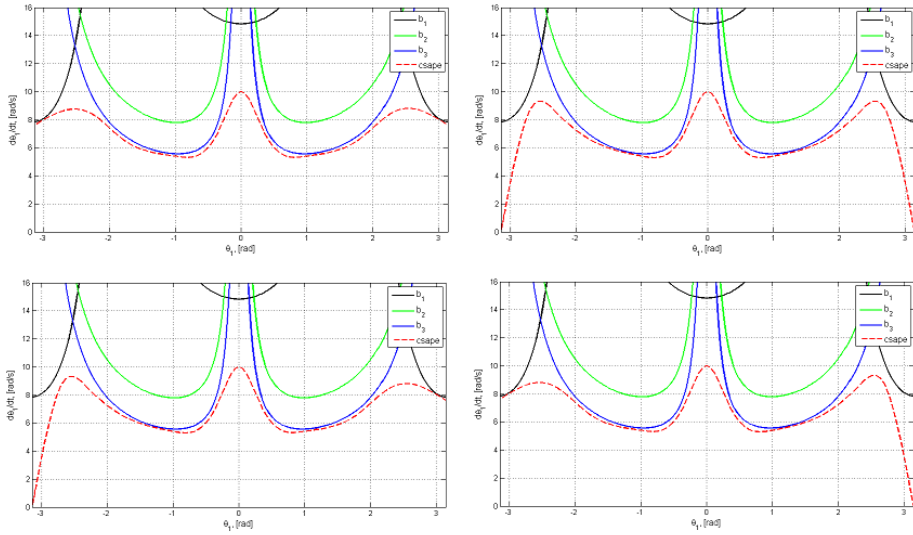


Figure 4.8: Showing planned velocity profiles for the horizontal circular case, with various end point settings, together with the velocity constraints. Red dashed line shows the chosen Spline. Black, blue and green lines show boundary lines for q_1 , q_2 and q_3 respectively.

Figure 4.8 shows the *csape* splines together with the boundary curves from Figure 4.6 for the various scenarios. The points used for the generation of the splines can be seen in Appendix A.6. We see that the *csape* function produces fitting splines for this set of boundary constraint curves.

Now, calculating the cost function of (4.29), it is possible to find the period of motion, T for the various scenarios. The calculations were done in Maple and the output of the calculation can be seen in Appendix B for the scenario with near time optimal end point velocities. The following periods of motion for the four different velocity profile scenarios were obtained:

Boundary level velocity at both the start and end point:

$$T_{f,\theta_1} = \int_{\theta_{1,0}}^{\theta_{1,e}} \frac{1}{\dot{\theta}_1} d\theta_1 = 0.9178941914 \approx 0.92 \text{ sec} \quad (4.33)$$

Zero velocity at both the start and end point:

$$T_{s,\theta_1} = \int_{\theta_{1,0}}^{\theta_{1,e}} \frac{1}{\dot{\theta}_1} d\theta_1 = 1.559323510 \approx 1.56 \text{ sec} \quad (4.34)$$

Zero velocity at the starting point, boundary level velocity at the end point:

$$T_{v,\theta_1} = \int_{\theta_{1,0}}^{\theta_{1,e}} \frac{1}{\dot{\theta}_1} d\theta_1 = 1.238609000 \approx 1.24 \text{ sec} \quad (4.35)$$

Boundary level velocity at the starting point, zero velocity at the end point:

$$T_{r,\theta_1} = \int_{\theta_{1,0}}^{\theta_{1,e}} \frac{1}{\dot{\theta}_1} d\theta_1 = 1.238609000 \approx 1.24 \text{ sec} \quad (4.36)$$

Note that T_{v,θ_1} and T_{r,θ_1} are planned with equal periods of motion, due to the symmetry around $\theta_1 = 0$ on the velocity profiles.

4.3.2 Case 2: Vertical Circular Motion

In this section the desired path will be a vertical circle. The vertical circle path can be parameterized as

$$\mathbf{P}_2^0 = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_{c2} \\ R_2 \sin(\theta_2) \\ R_2 \cos(\theta_2) + z_{c2} \end{bmatrix} \quad (4.37)$$

where θ_2 is the angle defining the circle, with radius $R_2 = R_1 = 0.2$ (m), centered on $(x_{c2} = 0.45, y_{c2} = 0, z_{c2} = 0.5)$ (m) in the yz -plane. We can then express the angular position as the path coordinate as follows

$$\theta_2 = \text{Atan2}(y(t) - y_{c2}, z(t) - z_{c2}) \quad (4.38)$$

The inverse kinematics can then be calculated analytically in a similar fashion as for the horizontal circle. Figure 4.9 illustrates the robot following the vertical circle path. The base joint can in this case be expressed as

$$q_1^*(\theta_2) = \arctan(R_2 \sin(\theta_2), x_{c2}), \quad (4.39)$$

From the figure we observe that the following two equalities must hold

$$l(\theta_2) + a_1 = \sqrt{x_{c2}^2 + R_2^2 \sin^2(\theta_2)} \quad (4.40)$$

and

$$h(\theta_2) + d_1 = R_2 \cos(\theta_2) + z_{c2} \quad (4.41)$$

With this we can express θ_m by using the law of cosines, and θ_b by using the law of sines

$$\theta_m = \arccos\left(\frac{a_3^2 + a_2^2 - l^2(\theta_2) - h^2(\theta_2)}{2a_3a_2}\right) \quad (4.42)$$

$$\theta_b = \arcsin\left(\frac{a_3 \sin(\theta_m)}{\sqrt{l^2(\theta_2) + h^2(\theta_2)}}\right) + \arctan\left(\frac{h(\theta_2)}{l(\theta_2)}\right) \quad (4.43)$$

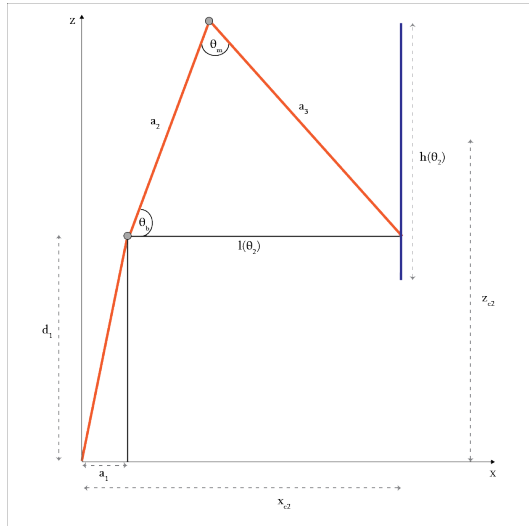


Figure 4.9: Geometrical view of the robot in the xz -plane. Blue line represents a circle in the yz -plane with radius R_2 , centered on (x_{c2}, y_{c2}, z_{c2}) .

The resulting expressions for the remaining joints are now dependant on a single variable, θ_2 .

$$q_2^*(\theta_2) = \frac{\pi}{2} - \theta_b \quad (4.44)$$

$$q_3^*(\theta_2) = \frac{\pi}{2} - \theta_m \quad (4.45)$$

In Figure 4.10 the joint angles q_i^* are shown as functions of $\theta_2 \in [-\pi, \pi]$. Note that q_1 starts and ends at 0, and that q_2 and q_3 starts in positions equal to the final position of the horizontal circle. This was achieved by choosing $R_2 = R_1$ and specifying the center of the vertical circle to be located directly above the position where the horizontal circle path ended, which will be useful in Chapter 6.

In Figure 4.11, we see the boundaries for the vertical circle path. Compared to the horizontal circle, these boundaries are less restrictive, meaning we can achieve higher velocity along the vertical circle than for the horizontal circle. This is because the third joint boundary curve is less restrictive in this case, while for the horizontal circle this boundary was the most restrictive. Figure 4.12 shows the velocity profile generated by three different cubic splines for the vertical circle with four different velocity profiles. The top left figure shows the result of using near time optimal velocity at the initial and final points. The top right figure shows the planning results when using zero velocity at both end points. The bottom left figure shows the planning result of using zero initial velocity and boundary level final velocity. Lastly, the bottom right figure shows the planning result of using boundary level initial

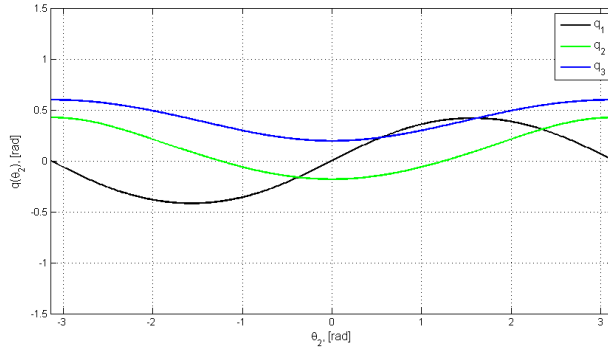


Figure 4.10: Showing q_i^* as functions of $\theta_2 \in [-\pi, \pi]$ for a vertical circle with radius R_2 , centered on (x_{c2}, y_{c2}, z_{c2}) .

velocity and zero final velocity. Although the benefit may be negligible, we see that the *csape* spline is the most aggressive of the splines, especially around $\theta_2 = -2.2$. The *csape* spline seems to be a good choice also in these scenarios, as was the case for the horizontal circle. In Figure 4.13 we see the velocity profile plotted with the boundary curves from Figure 4.11.

We can now calculate the cost function of (4.29), in the same manner as for the horizontal circle, using θ_2 instead of θ_1 . The periods of motion for the different velocity profiles were found to be

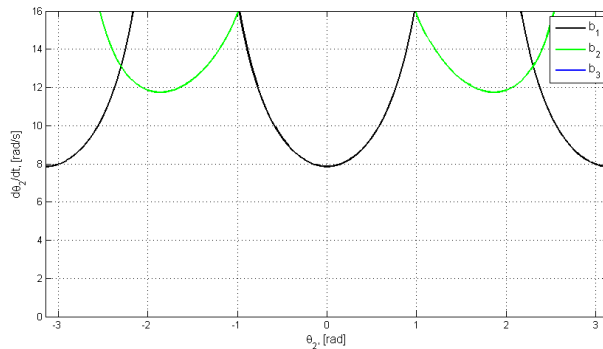


Figure 4.11: Boundary curves for the vertical circle path.

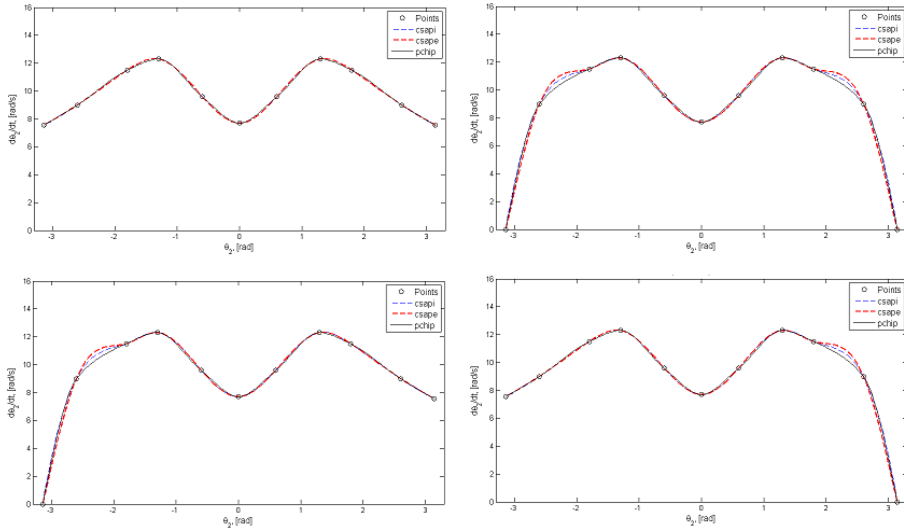


Figure 4.12: Comparing three types of cubic splines on four different velocity profile scenarios for the vertical circular path. *csapi* (blue dashed), *csape* (red dashed), *pchip* (black).

Boundary level velocity at both the start and end point:

$$T_{f,2} = \int_{\theta_{2,0}}^{\theta_{2,e}} \frac{1}{\dot{\theta}_2} d\theta_2 = 0.6407211965 \approx 0.64 \text{ sec} \quad (4.46)$$

Zero velocity at both the start and end point:

$$T_{s,2} = \int_{\theta_{2,0}}^{\theta_{2,e}} \frac{1}{\dot{\theta}_2} d\theta_2 = 1.454589090 \approx 1.46 \text{ sec} \quad (4.47)$$

Zero velocity at the starting point, boundary level velocity at the end point:

$$T_{s,2} = \int_{\theta_{2,0}}^{\theta_{2,e}} \frac{1}{\dot{\theta}_2} d\theta_2 = 1.047656910 \approx 1.05 \text{ sec} \quad (4.48)$$

Boundary level velocity at the starting point, zero velocity at the end point:

$$T_{s,2} = \int_{\theta_{2,0}}^{\theta_{2,e}} \frac{1}{\dot{\theta}_2} d\theta_2 = 1.047656910 \approx 1.05 \text{ sec} \quad (4.49)$$

We see that the planned periods of motion for the vertical circle are significantly faster than for the horizontal circle with the same radius.

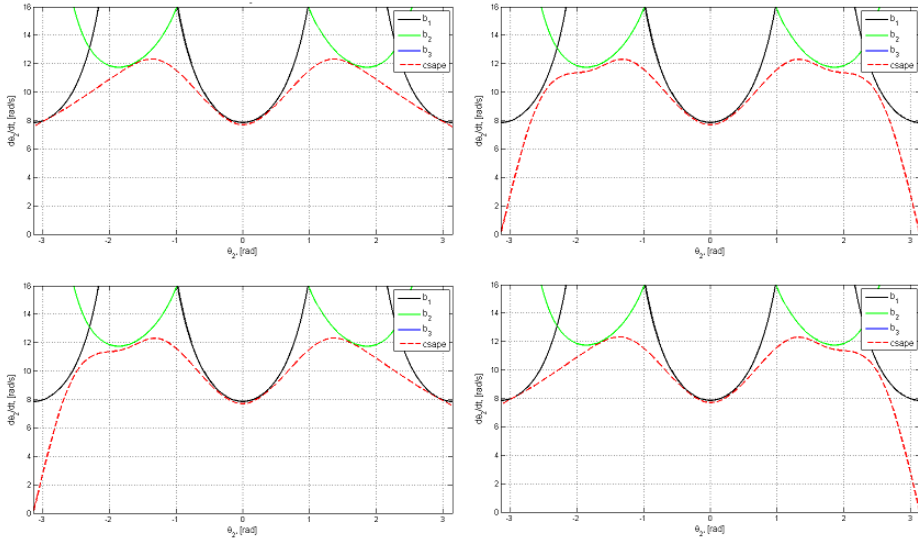


Figure 4.13: Showing planned velocity profiles for the vertical circular case, with various end point settings, together with the velocity constraints. Red dashed line shows the chosen Spline. Black, blue and green lines show boundary lines for q_1 , q_2 and q_3 respectively.

4.3.3 Case 3: Straight Line Motion

In the case of straight lines, the problem will be slightly different from the circular motions. For this paper, a straight line path was chosen to go up the radius of the horizontal circle, starting from the point (x_{c2}, y_{c2}, z_{c1}) and ending at (x_{c1}, y_{c1}, z_{c1}) . The x-coordinate works as a path coordinate, i.e. $x = x(t)$, meaning the velocity profile will be in m/s and not rad/s , as for the previous cases. The inverse kinematics will in this case be similar to the horizontal circle, only using the x-coordinate instead of θ_1 . For the base joint, the expression simply becomes

$$q_1^*(x) = 0 \quad (4.50)$$

since the base is not to be rotated throughout the path. Figure 4.14 illustrates the robot at the end of the path. Finding expressions for the second and third joints are then readily done as follows:

$$l(x) = \sqrt{(d_1 - z_{c3})^2 + (x - a_1)^2} \quad (4.51)$$

$$\theta_m = \arccos\left(\frac{a_3^2 + a_2^2 - l^2(x)}{2a_3a_2}\right) \quad (4.52)$$

$$\theta_f = \arctan\left(\frac{d_1 - z_{c1}}{x - a_1}\right) \quad (4.53)$$

$$\theta_b = \arcsin\left(\frac{a_3 \sin(\theta_m)}{l(x)}\right) - \theta_f \quad (4.54)$$

$$q_2^*(x) = \frac{\pi}{2} - \theta_b \quad (4.55)$$

$$q_3^*(x) = \frac{\pi}{2} - \theta_m \quad (4.56)$$

Figure 4.15 shows the joints throughout the path from $x = 0.45$ to $x = 0.65$. We can see that the base joint is kept steady at 0, while the second and third joint starts in the same position where the horizontal and vertical circle ended. The angle of the second joint, as seen by the green line, rises, while the third joint, seen by the blue line, decreases. The path ends before the joint values crosses the bounds of the manipulator workspace.

In Figure 4.16 the boundary curves for the velocity profile of the straight line path are plotted. We can see both boundary curves are kept relatively steady for the first half of the path, and then slowly decreasing for the second half. The boundary

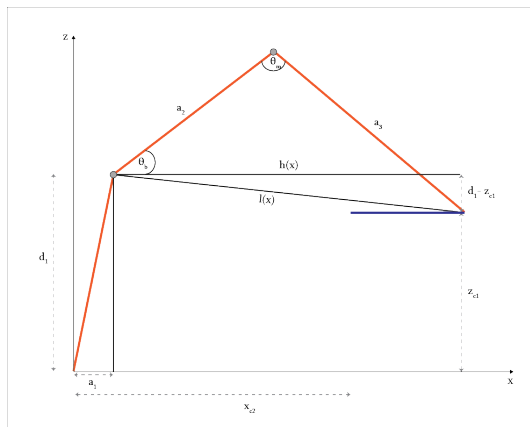


Figure 4.14: Geometrical view of the robot in the xz -plane. Blue line represents a straight line with length R_1 .

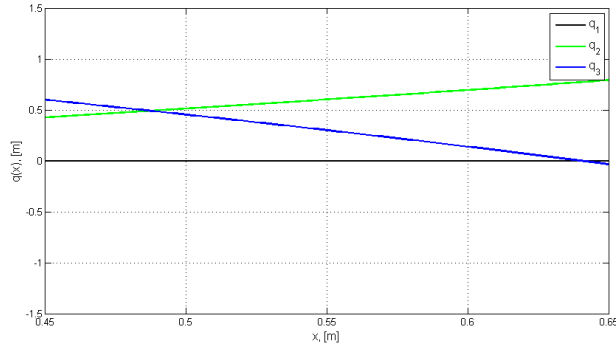


Figure 4.15: Showing q_i^* as functions of $x \in [0.45, 0.65]$, for the straight line path

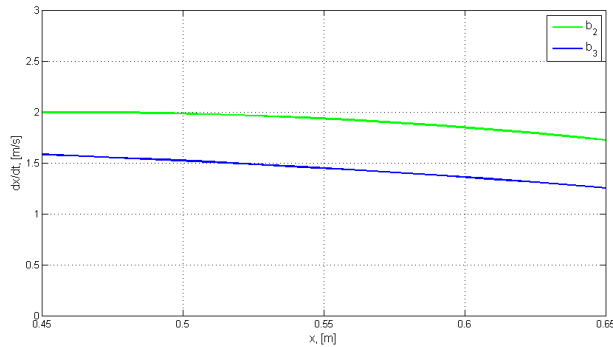


Figure 4.16: Boundary curves for the straight line path case.

generated by the third joint (blue line) is the most restrictive of the two, and the velocity profile curve will need to be kept below this boundary curve throughout the whole path. In Figure 4.17, the *csapi*, *csape* and *pchip* splines are plotted with for the same velocity profile scenarios as the two previous cases. We see that for this case, the *csapi* spline is not suited, for the velocity profile with zero initial velocity and boundary level final velocity. However, the *csape* spline is well suited for this situation. The velocity profiles are plotted with the boundary curves from Figure 4.16 in Figure 4.18.

Again, we can calculate the cost function, as for the previous cases, and find the planned periods of motions for the straight line path for the different velocity profile scenarios. This resulted in the following:

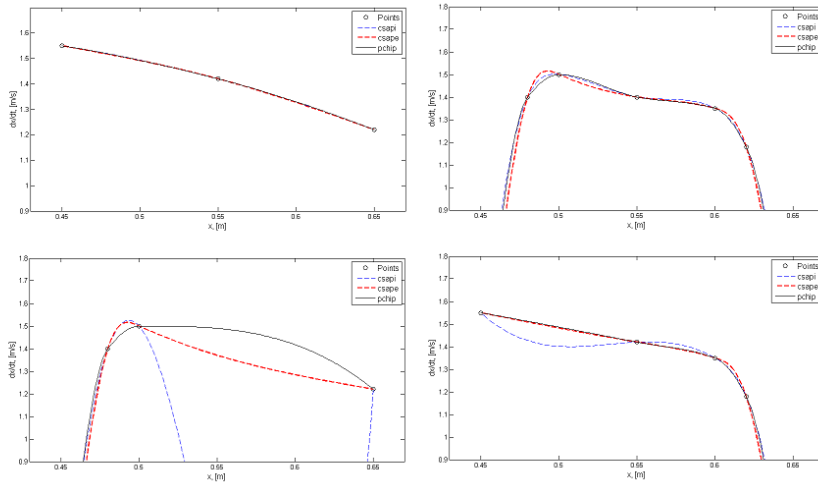


Figure 4.17: Comparing three types of cubic splines on four different velocity profile scenarios for the straight line path. *csapi* (blue dashed), *csape* (red dashed), *pchip* (black).

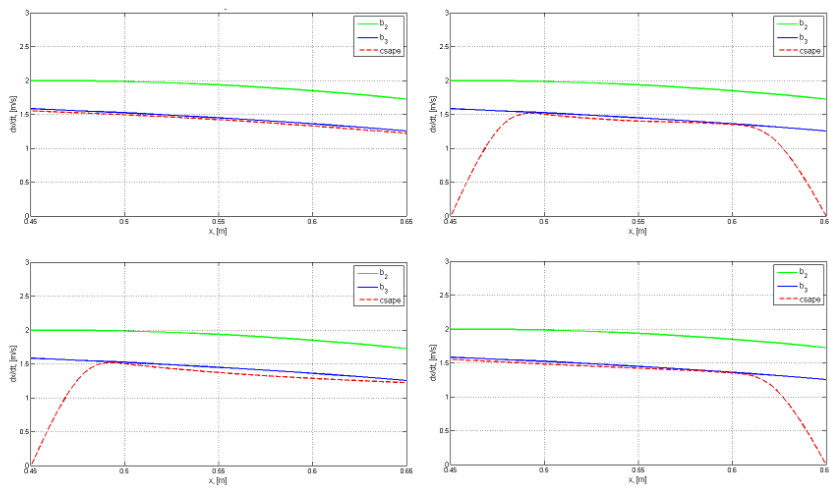


Figure 4.18: Showing planned velocity profiles for the straight line case, with various end point settings, together with the velocity constraints. Red dashed line shows the chosen Spline. Black, blue and green lines show boundary lines for q_1 , q_2 and q_3 respectively.

Boundary level velocity at both the start and end point:

$$T_{f,x} = \int_{x_0}^{x_e} \frac{1}{\dot{x}} dx = 0.1441163069 \approx 0.14 \text{ sec} \quad (4.57)$$

Zero velocity at both the start and end point:

$$T_{s,x} = \int_{x_0}^{x_e} \frac{1}{\dot{x}} dx = 0.3807147146 \approx 0.38 \text{ sec} \quad (4.58)$$

Zero velocity at the starting point, boundary level velocity at the end point:

$$T_{v,x} = \int_{x_0}^{x_e} \frac{1}{\dot{x}} dx = 0.2546097936 \approx 0.26 \text{ sec} \quad (4.59)$$

Boundary level velocity at the starting point, zero velocity at the end point:

$$T_{r,x} = \int_{x_0}^{x_e} \frac{1}{\dot{x}} dx = 0.2712823391 \approx 0.27 \text{ sec} \quad (4.60)$$

Note that for the straight line, the two planned periods of motion $T_{v,x}$ and $T_{r,x}$ are not equal. The two circle paths have symmetry around the midpoint on their velocity profiles, whereas the velocity profile for the straight line is asymmetrical.

4.4 Discussion

In this chapter we have performed motion planning for a set of case paths, and obtained the resulting planned periods of motion using different velocity profile scenarios. We have seen the results of velocity profile generations for three different interpolation functions in Matlab. The cubic spline function *csape* was able to handle velocity profile generation for both circular and straight line scenarios, making it a well rounded choice for velocity assignment tasks.

Having obtained the periods of motion, it is important to answer if these times are optimal. Clearly, the answer is no. The results from the motion planning presented in this chapter shows that we can only achieve near time optimal velocity profiles, using path constrained trajectory planning with cubic splines. It is probably possible to obtain even better results if one spends more time investigating the methods for generating the velocity profile, e.g. adding more points, or using other methods for curve fitting. The results obtained here are however expected to perform relatively well. From results found in the paper *Integrated Time-Optimal Trajectory Planning and Control Design for Industrial Robot Manipulator* presented by Pchelkin, Shiriaev, Robertsson and Freidovich [18], we know that our planned time of T_{f,θ_1} is at least 25% faster than what can be obtained using the commercial ABB planner running on maximum velocity.

It is also possible to attack the optimization problem by using the dynamic information obtained in Chapter 3.2, where we arrived at (2.17). This matrix equation can be rewritten to a non-matrix-form, which has convenient linear properties. The new representation for the differential equations takes the form

$$\sum_{i=1}^n m_{ij}(q)\ddot{q}_j + \sum_{j=1}^n \sum_{k=1}^n c_{ijk}(q)\dot{q}_j\dot{q}_k + g_i(q) = \tau_i \quad (4.61)$$

For trajectory generation, the linearity in $\dot{q}_j\dot{q}_k$ can be exploited, whereas for control problems the matrix form of (2.17) is better suited because of the non-singularity of $M(q)$, making it useful for manipulations of \ddot{q} . Now, substituting for the path coordinate, s , (2.15) becomes

$$\sum_{i=j}^n m_{ij}q(s)q'_j(s)\ddot{s} + \left(\sum_{j=1}^n \sum_{k=1}^n c_{ijk}q(s)q'_j(s)q'_k(s) + \sum_{j=1}^n m_{ij}q(s)q''_j(s) \right) \dot{s}^2 + g_i(q(s)) = \tau_i \quad (4.62)$$

This can now be stated in the compressed form

$$\alpha_i(s)\ddot{s} + \beta_i(s)\dot{s}^2 + \gamma_i(s) = \tau_i \quad (4.63)$$

Recall from Section 4.1, where a path parameterization was defined as $f(s)$. A suitable path parameter for the horizontal circle was found to be θ_1 . Now, having related the dynamics of the manipulator to a path coordinate, the dynamics along the path described in the previous section can be stated in terms of α_i , β_i and γ_i . Substituting into (4.63) we then get

$$\alpha_i(\theta_1)\ddot{\theta}_1 + \beta_i(\theta_1)\dot{\theta}_1^2 + \gamma_i(\theta_1) = \tau_i \quad (4.64)$$

where

$$\begin{aligned} \alpha_i(\theta_1) &= \sum_{i=j}^n m_{ij}q(\theta_1)q'_j(\theta_1)\ddot{\theta}_1 \\ \beta_i(\theta_1) &= \left(\sum_{j=1}^n \sum_{k=1}^n c_{ijk}q(\theta_1)q'_j(\theta_1)q'_k(\theta_1) + \sum_{j=1}^n m_{ij}q(\theta_1)q''_j(\theta_1) \right) \\ \gamma_i(\theta_1) &= g_i(q(\theta_1)) \end{aligned}$$

With this result, we see that we can define the optimization problem with τ being the constraining factor on the system, and thus optimization will be done with respect to the torque bounds. We can solve the time optimization problem using methods such as second-order cone programming or dynamic programming. For further discussion on these methods, see e.g. [24] and [21]. In Chapter 5 we will use the motion planning results from this chapter combined with motion control.

Chapter 5

Motion Control

In this chapter, the motion planning from Chapter 4 will be complimented with a feedback control design. We will investigate how the manipulator behaves when implementing a chosen motion control strategy. Simulations will be performed, where the manipulator is set to follow the desired trajectories obtained from the motion planning in Section 4.3. The obtained periods of motion from the simulations will be compared to the planned periods of motion, and we will analyze the performance of the controller in terms of tracking accuracy along the paths.

5.1 Orbital Stabilization

In Section 2.4 we looked at the classic PD tracking controller using feedback and gravity cancellation. For a deeper discussion on this control strategy, see [22]. In [16] an extension of this controller is introduced, guaranteeing tracking for rigid-joint robots. This is called PD+ control, and achieves global uniform asymptotic stability of the origin for the dynamic system on the form

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = \tau \quad (5.1)$$

which we found in Section 3.2 for the IRB140. The PD+ control law that is suggested in [16] takes the form

$$\tau = \hat{M}(q)\ddot{q}^*(t) + \hat{C}(q, \dot{q})\dot{q}^*(t) + \hat{G}(q) - K_p e - K_d \dot{e} \quad (5.2)$$

$$e = q - q^*, \quad \dot{e} = \dot{q} - \dot{q}^*(t) \quad (5.3)$$

where $\hat{M}(q)$, $\hat{C}(q, \dot{q})$ and $\hat{G}(q)$ are estimates for $M(q)$, $C(q, \dot{q})$ and $G(q)$ respectively. In [18] it is shown how to transform this regulator into one that is appropriate for orbital stabilization. This procedure is restated here.

First, let us define a new matrix $Q^*(t) = [q^*(t), \dot{q}^*(t), \ddot{q}^*(t)]$. Given an asymptotically stabilizing feedback control law, $\tau = U(e, \dot{e}, Q^*(t))$ where $q^*(t)$ is defined by

$$q = \begin{bmatrix} q_1^*(t) \\ q_2^*(t) \\ q_3^*(t) \end{bmatrix} = \Phi(s)|_{s=s^*(t)} = \begin{bmatrix} \phi_1(s) \\ \phi_2(s) \\ \phi_3(s) \end{bmatrix} \Big|_{s=s^*(t)} \quad (5.4)$$

and $s = s^*(t)$ is the solution to the differential equation $s = h(s)$, where the right hand side is obtained from the motion planning procedure from Chapter 4. The goal is to transform the previous control law into a new control law that ensures asymptotic orbital stability, meaning

$$\inf_{s \in [0, T]} \left\| \begin{bmatrix} q(0) - q^*(s) \\ \dot{q}(0) - \dot{q}^*(s) \end{bmatrix} \right\| \leq \delta \Rightarrow \left\| \begin{bmatrix} q(t) - q^*(s) \\ \dot{q}(t) - \dot{q}^*(s) \end{bmatrix} \right\| \leq \epsilon \quad (5.5)$$

$\forall \epsilon, \delta > 0$, where s is some variable that determines the shortest distance between the trajectories $q(t)$ and $q^*(t)$. Further one want to obtain asymptotic orbital closeness,

$$\lim_{t \rightarrow \infty} \left\{ \inf_{s \in [0, T]} \left\| \begin{bmatrix} q(t) - q^*(s) \\ \dot{q}(t) - \dot{q}^*(s) \end{bmatrix} \right\| \right\} = 0 \quad (5.6)$$

for a planned period T of the target trajectory. See [19, pp. 893-906] and [8, pp. 281-289] for related discussions.

Now, following [18], the propose is to substitute the classical tracking error, $e = q - q^*(t)$ and its time derivative to get

$$y = q - \Phi(s), \quad \dot{y} = \dot{q} - \Phi'(s)\dot{s} \quad (5.7)$$

$$s = Pr(q, \dot{q}), \quad \dot{s} = h(s) \quad (5.8)$$

where $Pr(q, \dot{q})$ is an arbitrary smooth projection operator defined to ensure that $Pr(q^*(t), \dot{q}^*(t)) = s^*(t)$. The new control law, $\tau = U(y, \dot{y}, Q^*(t))$, leading to asymptotic orbital stability then becomes

$$\tau = \hat{M}(q)\ddot{q}^*(t) + \hat{C}(q, \dot{q})\dot{q}^*(t) + \hat{G}(q) - K_p y - K_d \dot{y} \quad (5.9)$$

The stability statements of such control strategy are further discussed in [18], [19] and [8] and will not be restated here.

5.2 Motion Control Simulations on Case Trajectories

In this section the manipulator will be simulated following the case paths introduced in Chapter 4 with the regulator based on orbital stabilization introduced in the last section. The horizontal circle has a radius, $R_1 = 0.2$ (m), centered on the point $(x_{c1} = 0.65, y_{c1} = 0, z_{c1} = 0.3)$ in the xy -plane. The vertical circle also has a radius of

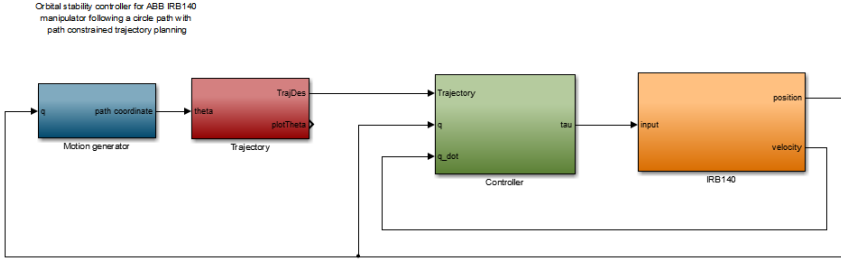


Figure 5.1: Simulink model of the IRB140 with orbital stabilization control and a motion planning structure.

$R_2 = 0.2$ (m) and is centered on $(x_{c2} = 0.45, y_{c2} = 0, z_{c2} = 0.5)$ in the yz -plane. The straight line will go up the radius of the horizontal circle, starting at (x_{c1}, y_{c1}, z_{c1}) and ending at (x_{c2}, y_{c2}, z_{c2}) . In addition to the current model of the robot dynamics, found in Section 3.2, a simple friction model was implemented,

$$F(v) = F_c \operatorname{sgn}(v) + \beta v \quad (5.10)$$

where F_c is the coulumb friction, β is the viscous friction coefficient and v is the relative velocity of the contact surfaces. The friction is compensated by adding to the control signal. The values of the friction parameters was given by Stepan Pchelkin from the work done in [18], and can be found in the Matlab models in the file folder related to this paper.

5.2.1 Simulink Layout

The complete dynamic model and control structure were implemented in Simulink, and the overall closed loop structure can be seen in Figure 5.1. The motion generator block contains a MATLAB function that generates the path coordinates θ_1, θ_2 and x , according to Section 4.3.1, Section 4.3.2 and Section 4.3.3 respectively, taking the position vector, q , as input. The trajectory block contains the results from the trajectory planning from Chapter 4, including cubic spline interpolations for generating the velocity profiles, $\dot{\theta}_1, \dot{\theta}_2$ and \dot{x} . Inverse kinematic relations were first computed in Maple and transformed to MATLAB code as seen in Appendix B. The output of this block is the desired trajectory for each path case, which for the horizontal circle will be $q^* = \Phi(\theta_1)$, as well as the derivatives $\dot{q}^* = \Phi'(\theta_1)\dot{\theta}_1$ and $\ddot{q}^* = \Phi''(\theta_1)\dot{\theta}_1^2 + \Phi'(\theta_1)\ddot{\theta}_1$, and similarly for the vertical circle and straight line. This signal is taken as input for the controller block, together with the measured signals of q and \dot{q} . In this block the control law is implemented, using estimates for the dynamic model together with the friction model. The gain matrices were tuned by

trial and error, and the friction model coefficients were obtained by the work done in [18]. The input signal, τ goes into the manipulator block, IRB140, where the robot is modeled by a Level 2-MATLAB S-function. Further details on this model can be seen in Appendix C.

5.2.2 Case 1: Horizontal Circular Motion

The model was simulated based on the trajectory planning for the horizontal circle from Section 4.3.1. Figure 5.2 shows the obtained time evolution of the function θ_1 from $-\pi$ to π . We see that the obtained period of motion is $T_{s,1} = 1.50$ seconds. The corresponding velocity profile can be seen in Figure 5.3, where we see that we have used the velocity profile with zero velocity at the end points. The same model was also simulated using the other velocity profiles presented in Section 4.3.1. The resulting time period when using boundary level velocities at the end points was found to be $T_{f,1} = 0.90$ seconds. The full set of planned and obtained time periods can be seen in Table 5.1 and Table 5.2 respectively. In Figure 5.4 we see the joint responses for the velocity profiles with zero and boundary level end point velocities. The continuous lines represent the obtained trajectories and the dashed lines represent the desired trajectories. Initial values for the joints are set equal to the initial positions from the desired trajectory. It can be seen that the tracking works well. The obtained trajectory align closely to the desired trajectory, especially for the velocity profile with zero velocity at the end points. A plot of the error vector,

$$Y = [y_1, y_2, y_3]$$

$$y_i = q_i - \phi_i(\theta_1), \quad i = 1, 2, 3$$

for the two velocity profile cases can be seen in Figure 5.6. We see that the

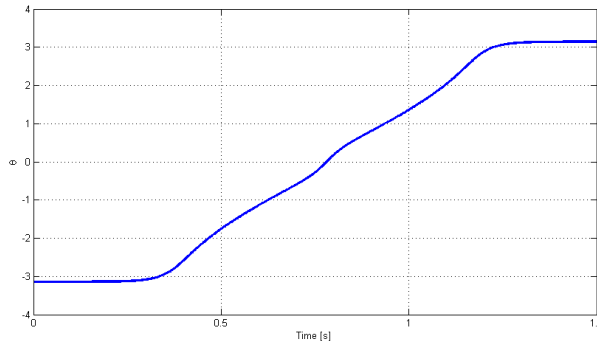


Figure 5.2: Time evolution of θ_1 for the velocity profile where starting and ending velocities are set to zero.

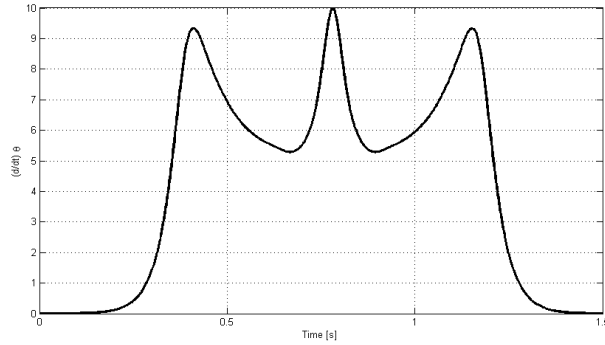


Figure 5.3: Obtained velocity profile, $\dot{\theta}_1$, generated by cubic spline interpolation for the horizontal circle path.

trajectory deviates from its desired trajectory by 0.0002 radians at its peak for the velocity profile with zero end point velocities. For the velocity profile with boundary level velocities we see a sudden spike to 0.035 radians deviation at the start of the motion. This is expected since the initial velocity is set to boundary levels. Figure 5.5 shows the trajectories in task space. The dashed line represent the desired task space trajectories as for the joint space plots. The difference between the two velocity profiles is also reflected here. We see a slightly larger error between the obtained trajectory and the desired trajectory for the velocity profile with boundary level end point velocity. This is most notable for $y(t)$. Also, it can be seen that the velocity profile with higher end point velocities reaches well over 1.5 orbits of the circle, whereas with zero end point velocities we reach only one full orbit in the same time span.

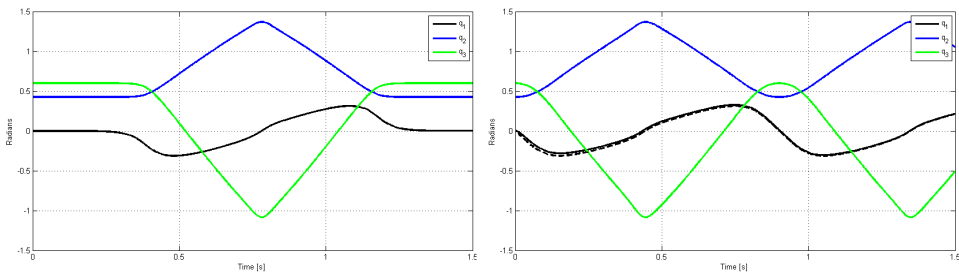


Figure 5.4: Showing obtained trajectory for the horizontal circle in joint space. q_1 (black), q_2 (blue), q_3 (green). Dashed lines show the desired trajectory. *Left figure:* Starting and ending velocity set to zero. *Right figure:* Starting and ending velocity set to boundary level.

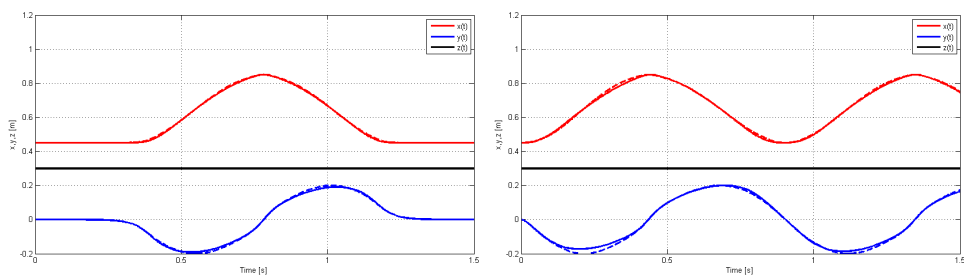


Figure 5.5: Showing obtained trajectory for the horizontal circle in task space. $x(t)$ (red), $y(t)$ (blue), $z(t)$ (black). Dashed lines show the desired trajectory. *Left figure:* Starting and ending velocity set to zero. *Right figure:* Starting and ending velocity set to boundary level.

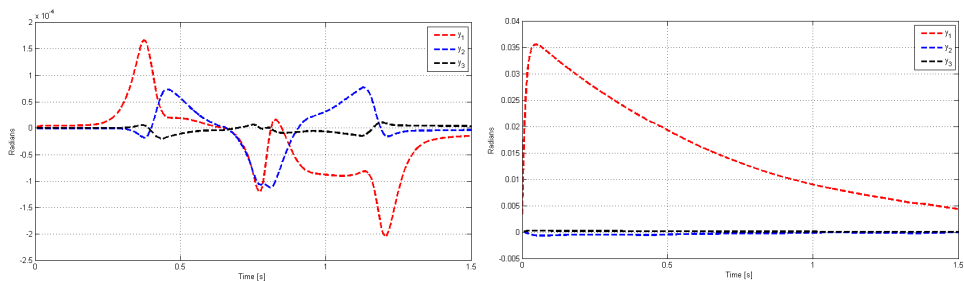


Figure 5.6: Showing the error vector for the horizontal circle path, Y_{hc} , i.e the error between the desired $\Phi(\theta_1)$ and obtained $q(t)$. y_1 (red), y_2 (blue), y_3 (black). *Left figure:* Starting and ending velocity set to zero. *Right figure:* Starting and ending velocity set to boundary level.

5.2.3 Case 2: Vertical Circular Motion

The simulation of the motion controller on the vertical circle case presented in Section 4.3.2 was done in a similar fashion as for the horizontal circle. The new path coordinate θ_2 , was generated by the motion generator block in Simulink (see Figure 5.1, and a new set of path functions, $\Phi(\theta_2)$ was implemented in the trajectory generation block. Figure 5.7 shows the obtained time evolution of $\theta_2 \in [-\pi, \pi]$, when using zero velocity end points, as seen in Figure 5.8. The total time of the motion was found to be $T_{s,2} = 1.34$ seconds, which is faster than the period of motion from the trajectory planning in Section 4.3.2, where we planned a time period of 1.46 seconds. Simulating with boundary level velocities at the end points, we obtained the period of motion, $T_{f,2} = 0.66$ seconds, compared to a planned time of 0.64 seconds. The resulting trajectory plots can be seen in Figure 5.9 for both cases, in the joint and task space. We observe that using boundary level velocity as initial velocity results in a larger deviation from the desired path during the first 0.5 seconds of the motion compared to the horizontal circle case. However, when using zero initial and ending velocity, the result is close to what was found for the horizontal circle case. This is reflected in Figure 5.10, where the error vector can be seen for both velocity profile cases. The zero velocity end point simulation resulted a deviation of around 0.0008 radians at its peak. For the case of boundary level velocities we see a similar situation as for the horizontal case, where the error is largest at the start of the motion, due to having high velocity at the initial point. This initial error decreases rapidly during the first 0.5 seconds of the motion, as reflected on the trajectory plots. In addition to the results presented here, the vertical circle was simulated using the velocity profile with zero starting velocity and boundary ending velocity, as well as the velocity profile with boundary starting velocity and zero ending velocity. The resulting periods of motion for these cases can be seen in Table 5.2.

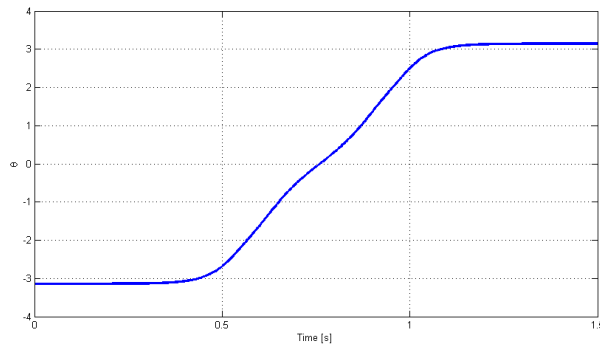


Figure 5.7: Time evolution of $\theta_2 \in [-\pi, \pi]$.

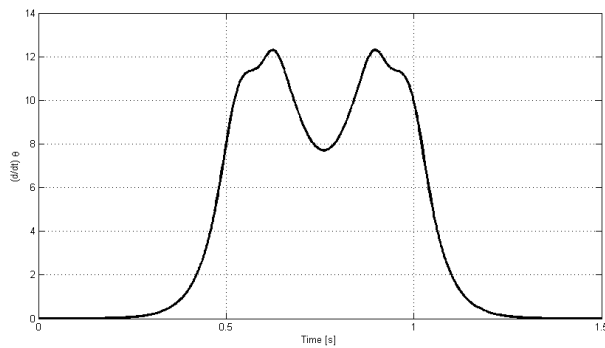


Figure 5.8: Resulting velocity profile for the vertical circle, obtained by cubic spline interpolation.

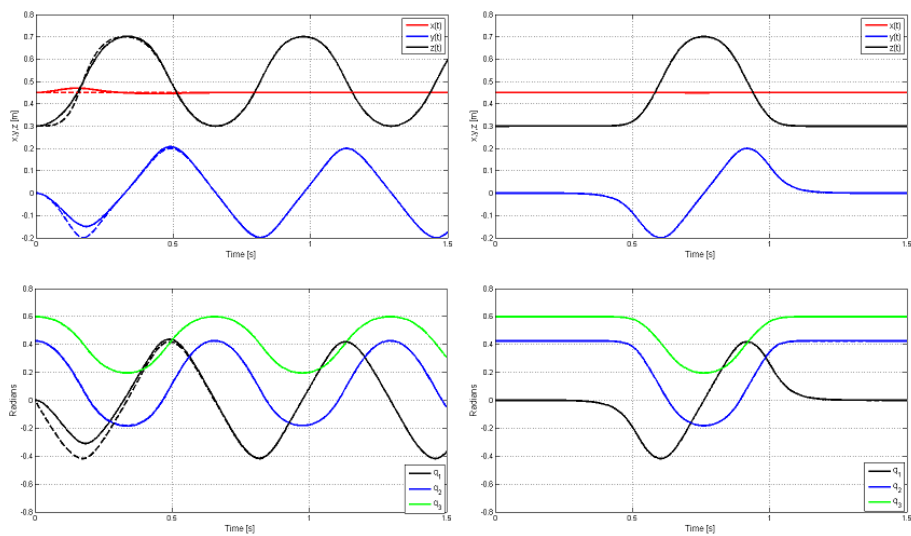


Figure 5.9: Showing the obtained (continuous) trajectories with the desired trajectories (dashed) from simulations for the vertical circular path. Upper figures show task space trajectories: $x(t)$ (red), $y(t)$ (blue), $z(t)$ (black). Lower figures show joint space trajectories: q_1 (black), q_2 (blue), q_3 (green). Figures to the left show the result of having boundary level velocities at the end points, while the figures to the right show the same result using zero end point velocities.

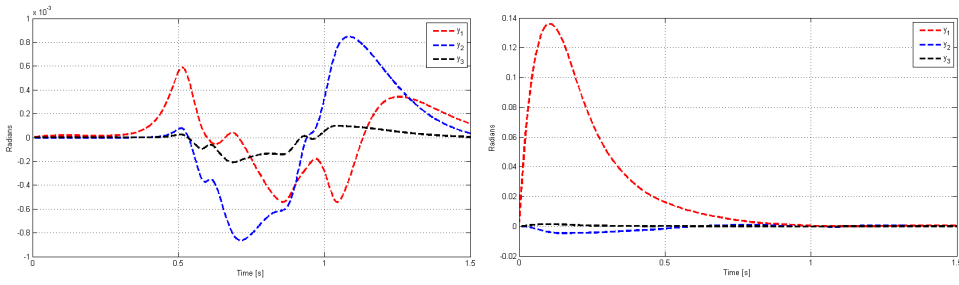


Figure 5.10: Showing the error vector for the vertical circle path, Y_{vc} , i.e the error between the desired $\Phi(\theta_2)$ and obtained $q(t)$. y_1 (red), y_2 (blue), y_3 (black). *Left figure:* Starting and ending velocity set to zero. *Right figure:* Starting and ending velocity set to boundary level.

5.2.4 Case 3: Straight Line Motion

In this section we will see the result of using the control structure for the straight line case. The path coordinate and desired trajectories were implemented to the model similarly to the previous cases. Figure 5.11 shows the resulting time evolution for the path coordinate, with initial value 0.45 (m) and final value 0.65 (m). The velocity profile can be seen in Figure 5.12, where we see that the end points are set to zero. This resulted in the period of motion $T_{s,3} = 0.36$ seconds, and the same simulation using boundary level end point velocities was found to be $T_{f,3} = 0.14$ seconds. Figure 5.13 shows the resulting trajectories in task space and joint space. Note that the resulting trajectories are plotted over different time spans. The results using boundary level end point velocities (left on figure), it can be seen that we get

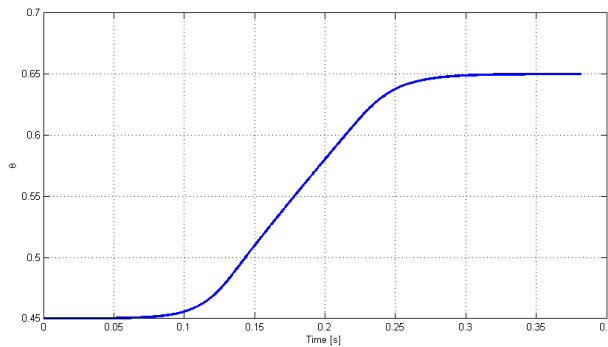


Figure 5.11: Time evolution for the straight line path with the velocity profile set to zero starting and ending velocity.

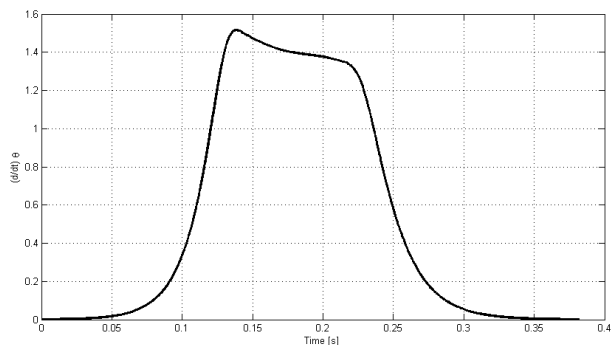


Figure 5.12: Resulting velocity profile with zero starting and ending velocity for the straight line path, obtained by cubic spline interpolation.

a slight, but stable deviation from the desired trajectory, while for the zero initial velocity case, the desired and obtained trajectories align closely. In Figure 5.14 we see the resulting error vector, Y for the two velocity profiles mentioned. We observe that the error is around 0.03 radians for the second joint (y_2), using boundary level end point velocities. This is less than for the circular paths. When using zero velocities at the end points we see a neglectable error of only 5×10^{-5} radians. The same simulation was also run using boundary initial velocity and zero final velocity, as well as a simulation with zero initial velocity and boundary final velocity. Resulting periods of motion, $T_{r,3}$ and $T_{v,3}$ can be seen in Table 5.2. The Matlab files used for the simulations can be found in the file folder related to this paper.

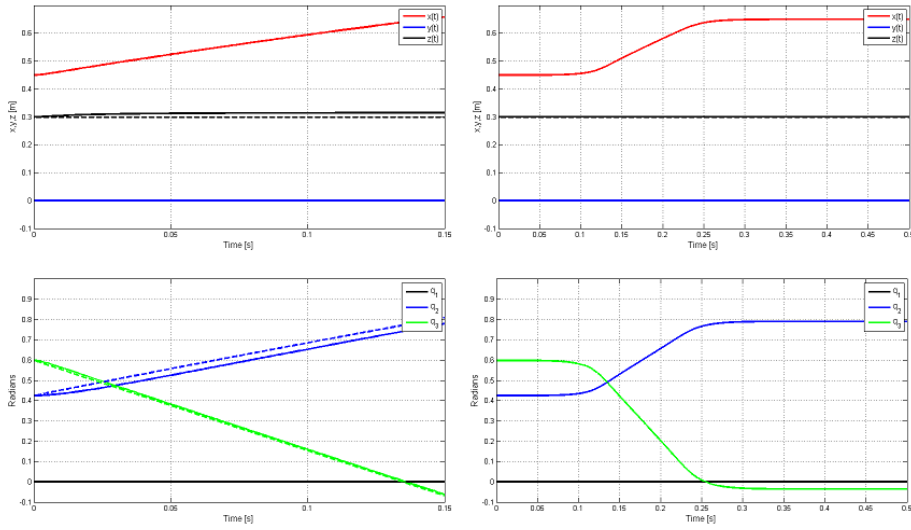


Figure 5.13: Showing the obtained (continuous) trajectories with the desired trajectories (dashed) from simulations for the straight line path. Upper figures show task space trajectories: $x(t)$ (red), $y(t)$ (blue), $z(t)$ (black). Lower figures show joint space trajectories: q_1 (black), q_2 (blue), q_3 (green). Figures to the left show the result of having boundary level velocities at the end points, while the figures to the right show the same result using zero end point velocities.

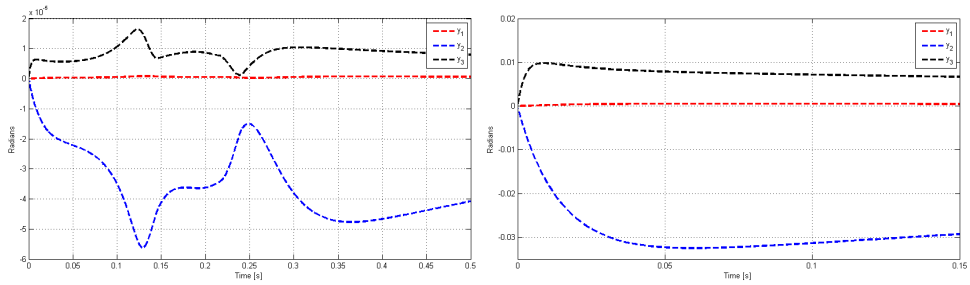


Figure 5.14: Showing the error vector for the straight line path, Y , i.e the error between the desired $\Phi(x)$ and obtained $q(t)$. y_1 (red), y_2 (blue), y_3 (black). *Left figure:* Starting and ending velocity set to zero. *Right figure:* Starting and ending velocity set to boundary level.

5.3 Discussion

Table 5.1 summarizes the planned periods of motion from Chapter 4. Table 5.2 shows the obtained periods of motion from simulations with motion control. We can see that the obtained values are close to the expected planned values, and in most cases the obtained periods of motion are slightly faster than the results from the trajectory planning. When analyzing the results for \dot{q} , it was observed that the joints are pushed close to the constraints found in Table 4.1. The resulting plots of \dot{q} can be seen by running the simulations with the Matlab files in the file folder related to this paper. As can be seen from the error vector plots, the tracking still performs well under these conditions. However, it may be beneficial to loosen up the velocity profile if one needs even better performance in terms of precision which is often the case in robotics. The simulations suggest that the method of path constrained trajectory planning combined with a feedback controller that achieves orbital stabilization works well for simple predefined paths such as the ones used here.

We have seen how the period of motion is significantly faster using boundary level initial values on the velocity profile. Using boundary level initial velocities might seem unnatural from what can be expected in practical situations. However, in the next chapter we will use these velocity profiles to connect the path cases and generate new combined paths. We have seen that the velocity profiles with boundary level initial velocity generates a starting error in the tracking of the desired trajectory. It would be interesting to see this error vector for simulations on connected paths where we have velocity through the connection point, using boundary level velocity for both the ending point of the first path segment and the initial point of the second path segment. With regards to time optimality in particular, the results from this chapter show that it would be desirable to avoid having to use zero velocity in the connection points between the path segments, if possible.

Table 5.1: Planned periods of motion, T , for various end point velocity profiles. $T_{f,i}$ shows the periods of motion for end point velocities set to boundary levels, $T_{s,i}$ shows the periods of motion for end point velocities set to zero, $T_{v,s}$ shows the periods of motion for zero initial velocity and boundary level final velocity, $T_{r,s}$ shows the periods of motion for boundary level initial velocity and zero final velocity.

Planned	$T_{f,s}$ (s)	$T_{s,s}$ (s)	$T_{v,s}$ (s)	$T_{r,s}$ (s)
Horizontal circle	0.92	1.56	1.24	1.24
Vertical circle	0.64	1.46	1.05	1.05
Straight line	0.14	0.38	0.26	0.27

Table 5.2: Obtained periods of motion, T , for various end point velocity profiles. $T_{f,i}$ shows the periods of motion for end point velocities set to boundary levels, $T_{s,i}$ shows the periods of motion for end point velocities set to zero, $T_{v,s}$ shows the periods of motion for zero initial velocity and boundary level final velocity, $T_{r,s}$ shows the periods of motion for boundary level initial velocity and zero final velocity.

Obtained	$T_{f,s}$ (s)	$T_{s,s}$ (s)	$T_{v,s}$ (s)	$T_{r,s}$ (s)
Horizontal circle	0.90	1.50	1.23	1.09
Vertical circle	0.66	1.34	1.08	1.04
Straight line	0.14	0.36	0.26	0.26

Chapter 6

Connecting Path Segments

In the previous chapter, the IRB140 manipulator was simulated following a set of trajectories with a control strategy implemented. In this chapter we want to analyze the possibility of connecting two or more paths, each with their own velocity assignment, and study the behavior of the manipulator for the desired trajectories with motion control.

6.1 The Usability of PCTP

The benefits of using PCTP (Path Constrained Trajectory Planning) have been seen in the previous chapters. However, the method has some limitations for some situations. In [10, pp 846-856], the usability of PCTP is discussed further. S. M. LaValle explains how some paths may be badly suited for path constrained trajectory planning. For example see the path in Figure 6.1. When the robot following the path reaches Point 1, as seen in the figure, the differentiability assumption will be violated and would require infinite acceleration to traverse while remaining on the path. However, some models may make it possible to make a complete stop at Point 1, and then start again. Consider a floating particle in the plane. The particle can be decelerated to rest exactly at Point 1, and then start in a new direction to exactly follow the curve. This assumes that the particle is fully actuated. [10, p. 850] If there are nonholonomic constraints, then the given path must at least satisfy them before accelerations can be considered.

At Point 2 in Figure 6.1, another interesting case can be seen. The robot might not be able to hold a high speed through this part of the path, if the corner is too sharp between the straight line and the curved line. Also, the curved line itself might bring difficulties in some cases, if the curvature is too high. However, it may be possible to hold (some) speed through point 2, in exchange for accuracy in the tracking. In the following sections we will investigate further on these issues, using the paths presented in the previous chapters. We will run simulations for the model of the IRB140 and connect the paths from the previous chapters, and analyze the results.

6.2 Simulations

The paths presented in Chapter 4, as well as some others, were combined to create longer paths consisting of two path segments. Several combinations were simulated using Simulink and Matlab, and some of the results are presented here. To be able to connect two paths, one needs to be able to change motion planning components between the path segments immediately after a path segment has completed. This was achieved by implementing a detection signal, which by default was set to 0, and then was given a value of 1 when a path segment was completed. The completion of the first path segment was detected by using a decrease detector which detects whenever a signal decreases. For example, for the horizontal circle, the path coordinate $\theta_1 \in [-\pi, \pi]$ will drop back down to $-\pi$ after reaching the value π , meaning the circular path segment is completed. From a programming point of view, it is straight forward to replace this method with equivalent solutions for cases where we do not want to traverse the full length of a path segment. We will see this in Chapter 7. Using the same sampling frequency as the standard ABB controller for the IRB140 ($\Delta_s = 0.004s$) means that we will overshoot the goal path by one sample value, which also needed to be handled for greater accuracy. When the detection signal is equal to 1, the path coordinate, velocity profile and desired joint trajectories, $\Phi(s)$, changes, and the new inputs get fed to the controller. For plotting the results, the detection signal also needed to be sent to the reference-generation mechanism. The Matlab and Simulink files can be found in the file folder related to this report.

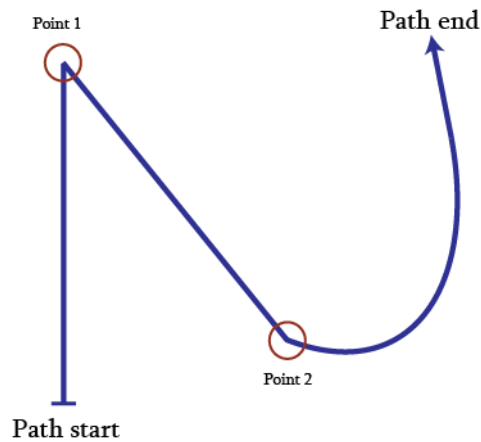


Figure 6.1: Example of a path badly suited for path constrained trajectory planning, similar to the example seen in [10].

6.2.1 Case 1: Horizontal Circle Connected to a Straight Line

The first case we will look at is a horizontal circle followed by a straight line. We will use the circle and straight line presented in Chapter 4. This means a circle with a radius, $R_1 = 0.2$ (m), centered on $(x_{c1} = 0.65, y_{c1} = 0, z_{c1} = 0.3)$. (m) The straight line starts at $(x_{c2} = 0.45, y_{c1} = 0, z_{c1} = 0.3)$ (m) and goes up the radius of the horizontal circle, ending at the circle center. We see that we have generated a 90 degree corner connection point between the circle and straight line. In Chapter 4 we planned trajectories using velocity profiles with zero starting and ending velocities as well as boundary level starting and ending velocities for both the circle and the straight line. In Figure 6.2 the resulting trajectory from the simulation of the combined motion is plotted in the joint space. The starting and ending velocities are set to zero for both path segments. The total time period of this motion was found to be $T_{hcs1,s} = 1.728$ (s). If we compare this time period to the planned period of $T_{s,1} + T_{s,3} = 1.940$ (s), we see that the result from the simulation is faster than the combined planned period of motion for the two cases. We see that the dashed line, which show the desired trajectory, line up very close to the actual trajectory shown by the continuous lines. In Figure 6.3 the same result can be seen in task space. We can see a minor error for $z(t)$ when the path reaches the straight line. This error is reflected in the joint space error plot, seen in Figure 6.4, where we can see an error spike of 0.0022 radians for $y_2 = q_2 - \phi_2(P(q))$. In Figure 6.5 we see the same simulation, using boundary level velocities for the velocity profiles through the connection point between the horizontal circle and the straight line. We

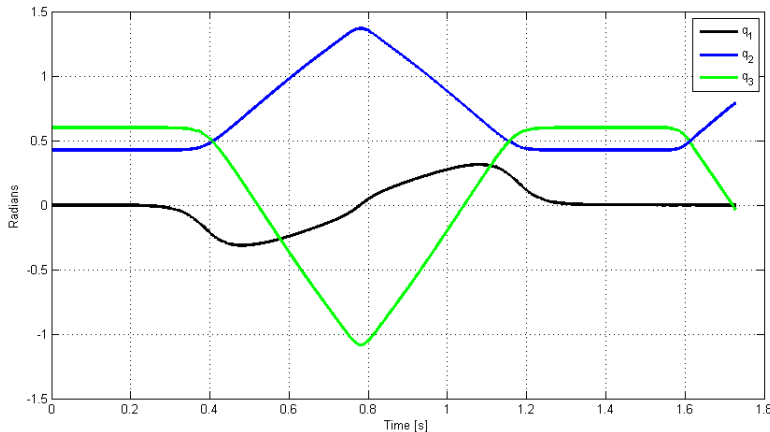


Figure 6.2: Resulting trajectory in joint space of combining a horizontal circle and a straight line path. The velocity profiles are set with low starting and ending velocities for both path segments. Dashed lines show desired trajectory.

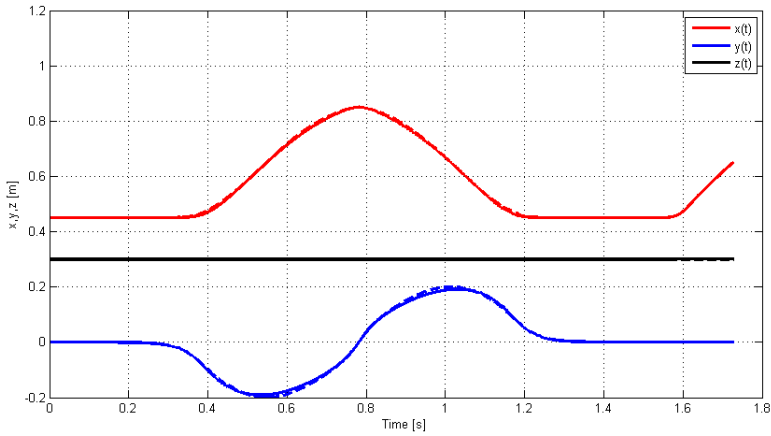


Figure 6.3: Resulting trajectory in task space of combining a horizontal circle and a straight line path. The velocity profiles are set with low starting and ending velocities for both path segments. Dashed lines show desired trajectory.

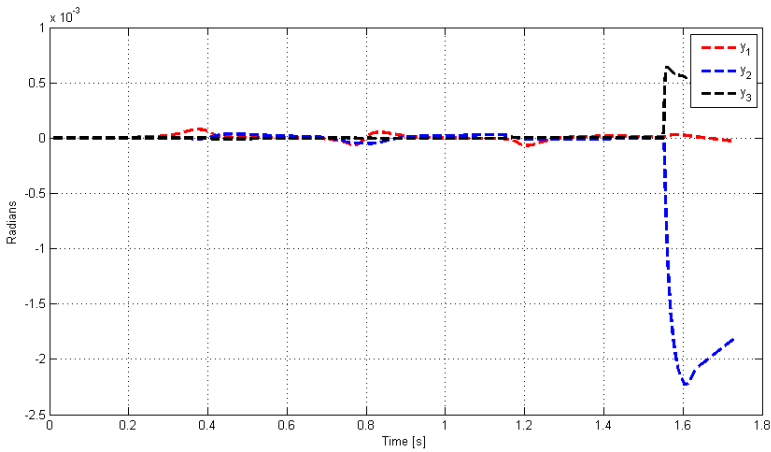


Figure 6.4: Error between desired and obtained joint values on the combined horizontal circle and straight line path.

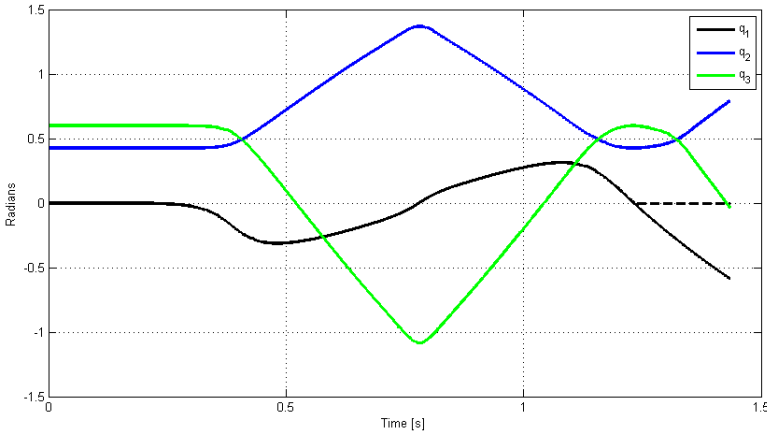


Figure 6.5: Resulting trajectory in joint space of combining a horizontal circle and a straight line path. The velocity profiles are set with boundary level velocities at the connection point for both path segments. Dashed lines show desired trajectory.

see that the first joint, q_1 , is unable to follow the desired trajectory through this point and continues with a circular motion instead. The second and third joints are, however, able to follow their desired trajectories through this point with only minor errors, as the desired trajectories for these joints makes for natural soft connections between the two path segments. The time period of motion with these velocity settings was found to be $T_{hcsl, vr} = 1.434(\text{s})$, which is faster than the planned time of $T_{v,1} + T_{r,3} = 1.509(\text{s})$.

6.2.2 Case 2: Vertical Circle Connected to a Straight Line

In this section we will be using a vertical circle path instead of the horizontal circle from Case 1, combined with the straight line path. The vertical circle will be the same as the one presented in Chapter 4, i.e. $R_2 = R_1 = 0.2\text{m}$, centered around $(x_{c2} = 0.45, y_{c2} = 0, z_{c2} = 0.5)$. The trajectory will be a full circular motion around the vertical circle followed by the horizontal straight line, corresponding to the straight line from the previous case.

The resulting trajectory from simulations, using zero initial and ending velocities on the velocity profiles is seen in Figure 6.6 for the joint space, and Figure 6.7 for the task space. We see in Figure 6.8 that the trajectory is followed with about the same accuracy as in case 1, showing how velocity assignment with PCTP is suited for corner paths when we are able to stop at the corner and then start accelerating in another

direction. The total time of the combined path was found to be $T_{vcsl,s} = 1.636(s)$, whereas the combined planned time from Chapter 4 was $T_{s,2} + T_{s,3} = 1.835(s)$.

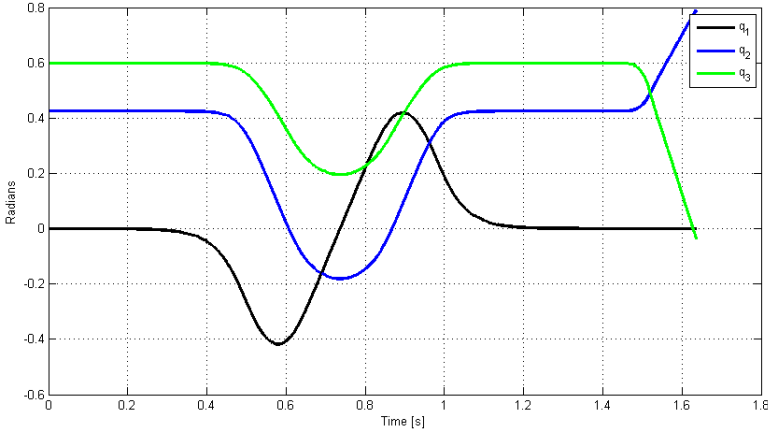


Figure 6.6: Resulting trajectory in joint space of combining a vertical circle and a straight line path. The velocity profiles are set with low starting and ending velocities for both path segments. Dashed lines show desired trajectory.

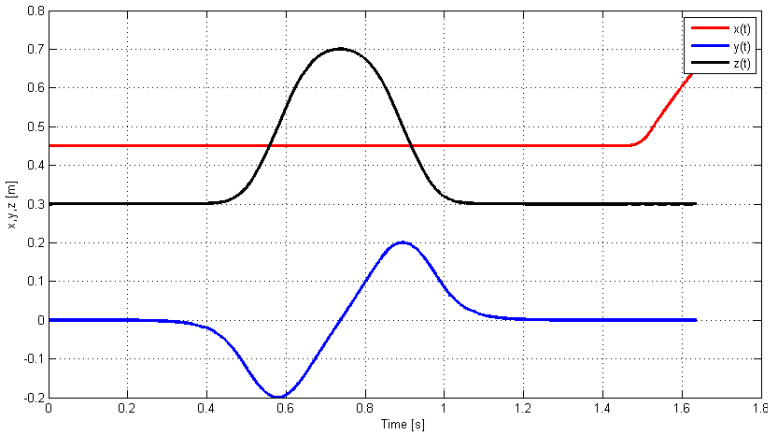


Figure 6.7: Resulting trajectory in task space of combining a vertical circle and a straight line path. The velocity profiles are set with low starting and ending velocities for both path segments. Dashed lines show desired trajectory.

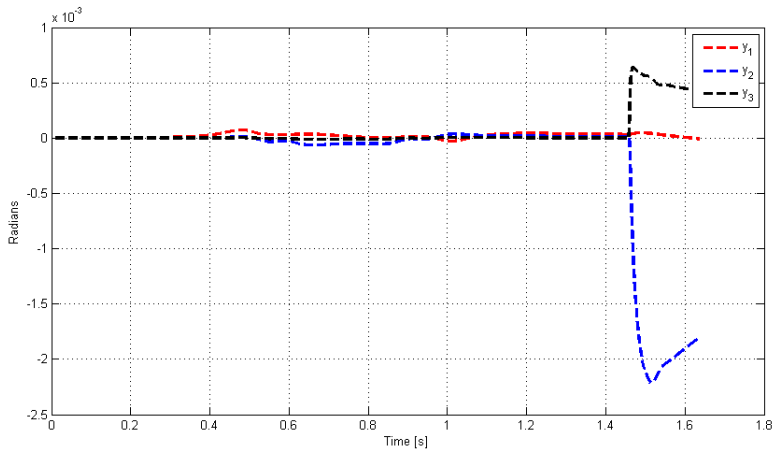


Figure 6.8: Error between desired and obtained joint values on the combined vertical circle and straight line path.

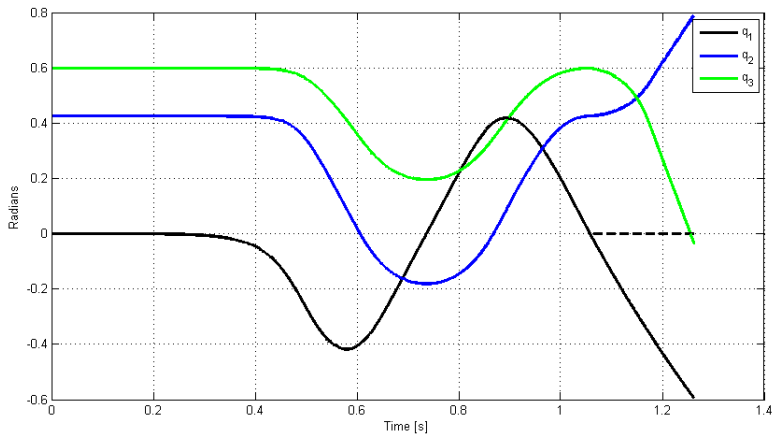


Figure 6.9: Resulting trajectory in joint space of combining a vertical circle and a straight line path. The velocity profiles are set with boundary level velocities at the connection point for both path segments. Dashed lines show desired trajectory.

In Figure 6.9 we see the joint space result of the same simulation performed with velocity profiles set to boundary level velocities through the connection point for both path segments. Again we see that the first joint, q_1 , is unable to follow the desired trajectory, while the second and third joint, q_2 and q_3 respectively, are able to follow their trajectories. It is obvious that sharp corners, such as 90 degrees in this case, is problematic for PCTP when we want to keep the velocity during the connections, as expected. The total time when using boundary level velocity values for the starting and ending point was found to be $T_{vcsl, vr} = 1.262(\text{s})$ for the vertical circle combined with the straight line. In comparison the combined planned time period from the planning phase can be found as $T_{v,2} + T_{r,3} = 1.319(\text{s})$

6.2.3 Case 3: Horizontal Circle Connected to a Vertical Circle

As our last case we will combine the horizontal circle from Case 1 with the vertical circle from Case 2 to obtain a new path. The trajectory will start by traversing the horizontal circle once, and then immediately traverse the vertical circle. The two earlier cases had 90 degree corner connection points, and we saw how tracking of the desired trajectories failed when using boundary level velocities through the connection points. In this case the direction of the trajectories before and after the connection point are equal. This trajectory will give a softer connection point, and it is interesting to see if it is possible to have velocity through the connection for this case. First, manipulator was simulated for the trajectory using zero starting and ending velocities for both velocity profiles. The resulting joint space and task space trajectory can be seen in the upper figures of Figure 6.10 and Figure 6.11 respectively.

We see that the desired (dashed line) and obtained (continuous) trajectories align closely throughout the entire path, and it is difficult to separate the two from the figure without zooming in. In the upper figure of Figure 6.12 we see that the error in this case reaches only 0.0002 radians at its heighest, giving a better performance than for Case 1 and 2. The total time period of the combined motion was found to be $T_{hvc, s} = 2.909(\text{s})$ for zero end point velocities for both path segments. In comparison the planned time can be found as $T_{s,1} + T_{s,2} = 3.014(\text{s})$.

The lower figures of Figure 6.10 and Figure 6.11 shows the resulting trajectory for joint and task space respectively, when using boundary level velocity profiles through the connection point for the horizontal and vertical circles. We see that in this case, the joints are able to follow their desired trajectories even when using (sub-)optimal velocities through the connection point. The lower figure in Figure 6.12 shows a deviation spike of ≈ 0.0045 radians at the connection point in this case. By this, we can conclude that using PCTP is suitable for combining preplanned path segments when the segments take a form which are naturally suited for connection.

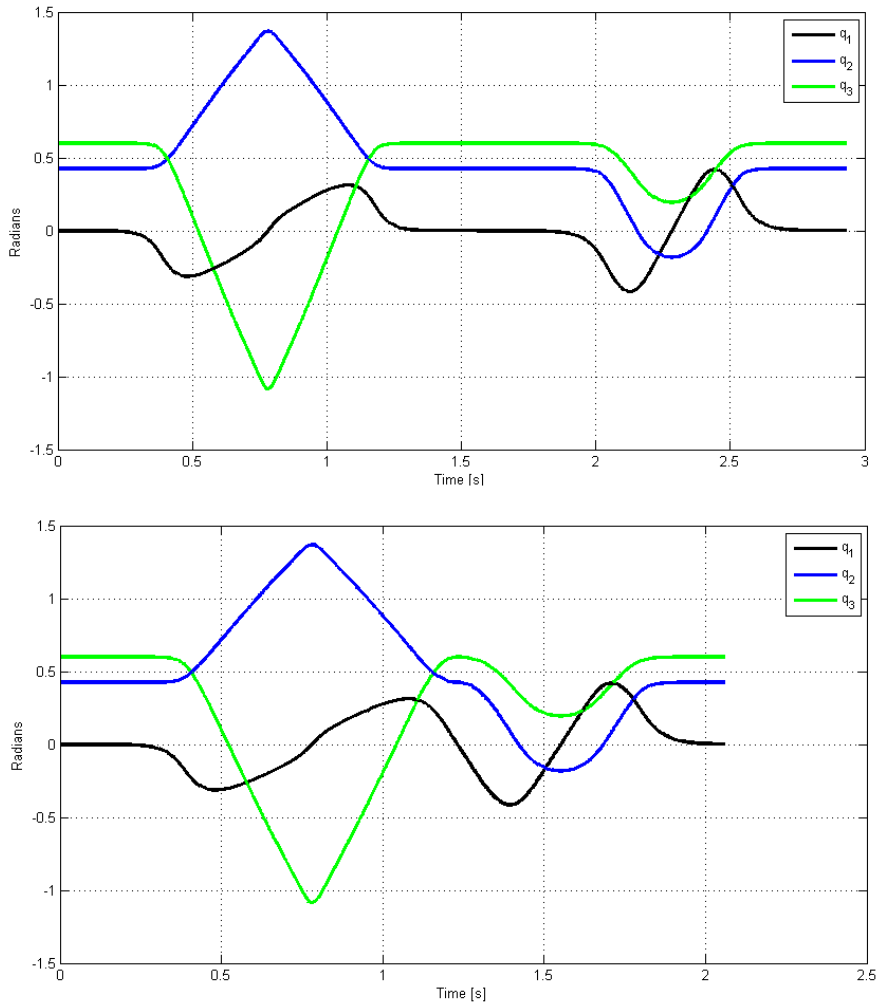


Figure 6.10: Resulting trajectory in joint space of combining a horizontal circle and a vertical circle path. *Upper figure:* Velocity profiles are set to zero at the connection point between the two paths. *Lower figure:* Velocity profiles are set to boundary level velocity through the connection point. Dashed lines show desired trajectory.

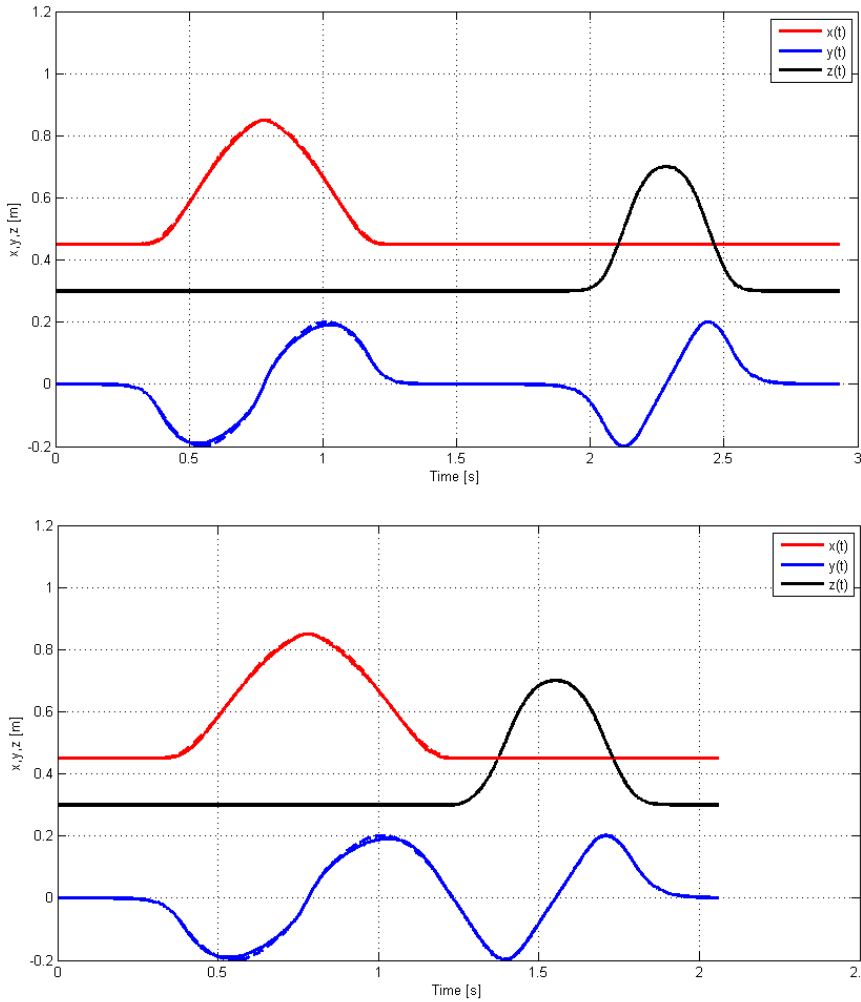


Figure 6.11: Resulting trajectory in task space of combining a horizontal circle and a vertical circle path. *Upper figure:* Velocity profiles are set to zero at the connection point between the two paths. *Lower figure:* Velocity profiles are set to boundary level velocity through the connection point. Dashed lines show desired trajectory.

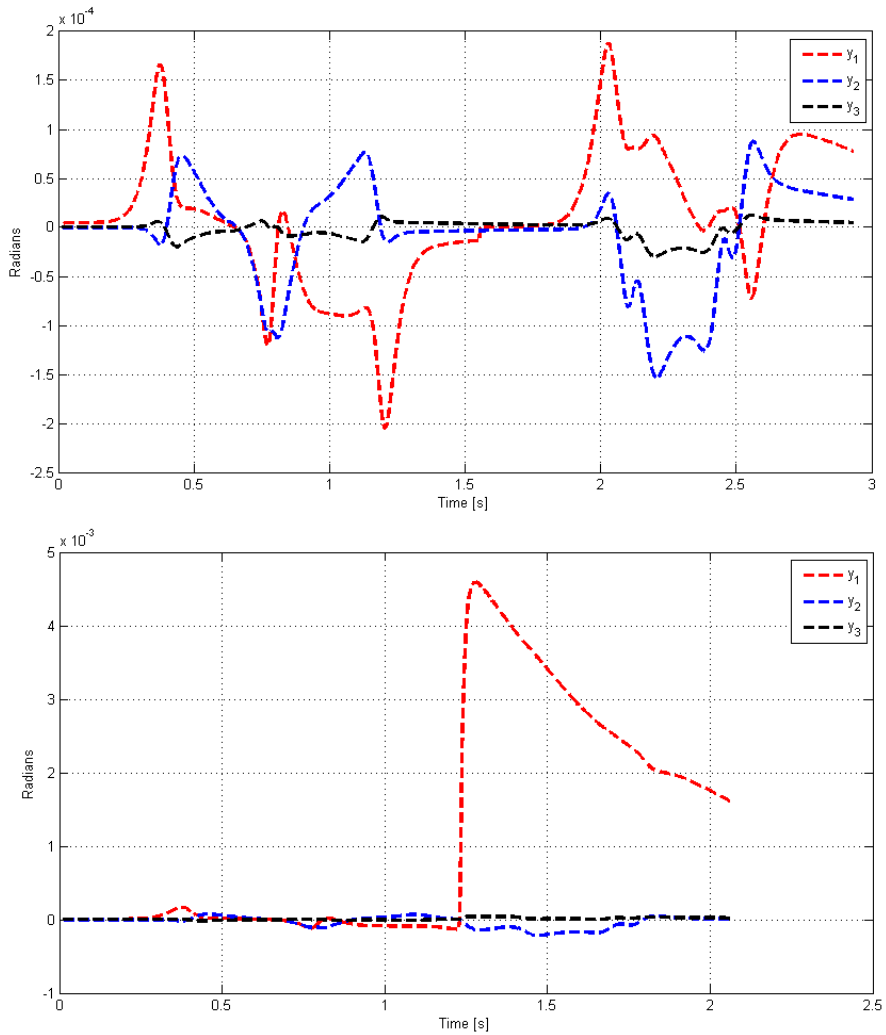


Figure 6.12: Resulting error vector when combining a horizontal circle and a vertical circle path. *Upper figure:* Velocity profiles are set to zero at the connection point between the two paths. *Lower figure:* Velocity profiles are set to boundary level velocity through the connection point.

The total time period of the motion in this case was found to be $T_{hvc,vr} = 2.057(\text{s})$, whereas the combined planned time period can be found as $T_{v,1} + T_{r,2} = 2.286(\text{s})$.

6.3 Discussion

The results of this chapter highlights the points given by S. M. LaValle in [10, pp 846-856], where the usability of path constrained trajectory planning is discussed. We see from the simulation results that sharp corner points are troublesome when applying velocity through the connection between path segments. This results were as expected. It was, however, seen that the method works well when being able to decelerate to a full stop at the corner point and accelerate in a new direction.

Comparing the periods of motion obtained in this chapter, we have seen that there is potential for significantly improving the time periods using higher velocities at the connection points. An interesting observation was seen for scenarios where the geometry of the path segments are well conditioned to make soft connection points (i.e. no corners). This was demonstrated in Case 3, where we connected a horizontal circle to a vertical circle. For this case, we were able to assign boundary level velocity at the connection point for both path segments, resulting in a maximum deviation spike of 0.0045 radians from the desired trajectory. It is suggested that this observation can be used to redesign sharp corner points into softer connections in exchange for some path accuracy error. This will be investigated in Chapter 7 for the horizontal circle path connected to a straight line.

Chapter 7

Connections with Sharp Corners

In the previous chapter we simulated the IRB140 manipulator for various paths consisting of two connected path segments. We could see how sharp corner connection points can cause the manipulator to fail to track the desired path completely when having a velocity profile with high velocity through the connection. This is clearly not an acceptable situation for practical purposes, meaning the velocity profile needs to be set to lower velocity, or even a full stop at the corner point to be able to traverse the corner with a high level of precision. However, there are scenarios where time optimality is given a higher priority than precision, and for such cases, it would be desirable to be able to achieve an acceptable amount of precision while having time optimization as our primary goal. This chapter deals with this problem, and will be based on Case 1 from the previous chapter, where a horizontal circle was connected to a straight line, making a corner with an angle of 90 degrees.

7.1 Corners in Pick-and-Place Motions

As briefly mentioned in Section 2.3, point-to-point motions are motions where a high level of freedom exists on the path between two locations. Algorithms dealing with such motions are given a pick position and place position (as well as intermediate positions to avoid collisions during the motion), and the task is to travel as fast as possible between these two points. This means that the manipulator moves as fast as possible from an initial configuration to a final configuration. The trajectory needs to be respecting certain constraints while fulfilling a performance criterium. The most common criterium in industry, where the goal is higher production volume, is the time optimality criterium. A method for obtaining smooth motions between two points is presented in [11, pp. 42-52]. Our main focus in this paper is path constrained motion, so we will not go into further detail on pick-and-place motion here, however there is some inspiration to be found in the way of dealing with corners in point-to-point motion. In [9] the problem of traversing corners is discussed for pick-and-place motions (i.e linear segments). Figure 7.1 shows a path consisting

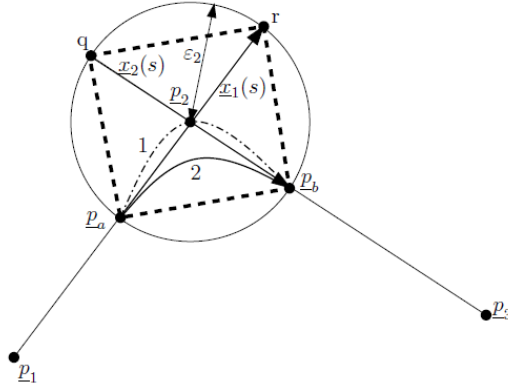


Figure 7.1: Illustration of the blending procedure developed by Lloyd and Hayward [9]. Figure from [23].

of straight lines between 3 points, p_1 , p_2 and p_3 . A so-called blend function, $x(t)$ is needed to describe the movement between the two linear segments. The circle around p_2 illustrates a blend sphere. The blend sphere is specified in which the path may deviate from the linear segments, i.e the blending radii defined at each point is given as $\epsilon_1 = \epsilon_3 = 0$ and $\epsilon_2 > 0$. The blend start and end points, p_a and p_b are then defined by

$$p_a = p_2 - \epsilon_2 \frac{p_2 - p_1}{\|p_2 - p_1\|} \quad (7.1)$$

$$p_b = p_2 + \epsilon_2 \frac{p_3 - p_2}{\|p_3 - p_2\|} \quad (7.2)$$

At these points, a set of boundary conditions must hold in order to guarantee continuity of the geometric path.

$$x(t = t_1) = p_a, \quad x(t = t_2) = p_b, \quad (7.3)$$

$$\dot{x}(t = t_1) = v_a, \quad \dot{x}(t = t_2) = v_b, \quad (7.4)$$

$$\ddot{x}(t = t_1) = a_a, \quad \ddot{x}(t = t_2) = a_b, \quad (7.5)$$

These six conditions can be satisfied by connecting the blend start and end points by a fifth-order polynomial. To see more details on this blending procedure, see [9].

7.2 Sharp Corners in Path Constrained Motion

Traversing corners with high velocities for path constrained motion does not seem to be discussed much in literature. The blending procedure from the previous section is designed for linear segments between a set of points. For path constrained motion,

this will usually not be the case, and so we have to look for other solutions. There was no solution found to this problem for path constrained motions in the literature study, so an attempt to deal with the problem is presented here. Note that much of the literature on the subject of path constrained trajectory planning use a path coordinate, s , with a domain starting point at $s_0 = 0$ and ending point at $s_e = 1$, while in this paper we have used more specific values, e.g. a path coordinate, θ_1 , starting at $\theta_{1,0} = -\pi$ and ending at $\theta_{1,e} = \pi$ for circular motion.

Section 7.1 explains for pick-and-place motions how a sphere can be defined such that we are allowed to round the corner anywhere within this sphere. The results from Chapter 6 shows for path constrained motion how it is possible to traverse a corner point with boundary level velocity assignments when the geometry of the path segments are well conditioned. These two aspects can be combined to generate a possible solution to corners for path constrained motion. In Figure 7.2 we see an illustration of two path segments, each having their own path coordinate, θ_1 and θ_2 with corresponding velocity profiles $\dot{\theta}_1$ and $\dot{\theta}_2$. The connection point between the two path segments will be a sharp corner. The transition segment will be a new path segment around the corner point, with the objective of making us able to approximately track the corner point without having to deaccelerate to zero velocity during the motion. This will come at the cost of some deviation error from the desired path, due to rounding of corners. The method can be exemplified for the horizontal circular path segment connected to a straight line path segment from the circle circumference in to the circle center point. This is equal to Case 1 from the previous chapter. (See Section 6.2.1) Figure 7.3 shows the desired path (blue) around the corner point. The circle (red) shown in the figure is going to be used to generate a transition path segment.

Consider the path coordinate for the horizontal circular path, $\theta_1 \in [-\pi, \pi]$ from the previous chapters. Let us change the domain of the path coordinate to $\theta_1 \in [-\pi, \beta]$, where

$$\beta = \pi - \alpha \quad (7.6)$$

The angle α can be seen in Figure 7.3. We see that this angle can be derived as

$$\alpha = \arctan\left(\frac{r_t}{R_1}\right) \quad (7.7)$$

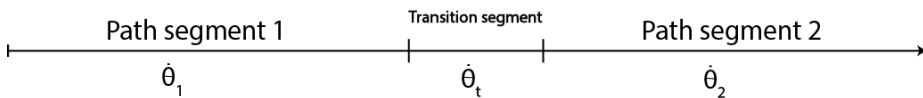


Figure 7.2: Combining two path segments using a third segment designed as a suitable transition path.

where R_1 is the radius generated by making the curved blue line into a full circle, and r_t is the radius of the transition circle. This means that the path gets reduced by the length

$$L = \frac{\alpha\pi R_1}{180} + r_t \quad (7.8)$$

In Figure 7.4 we see the general geometry of a circle segment. In our case, the length h can be found as

$$h = R_1 \left(1 - \cos\left(\frac{\theta}{2}\right) \right) = R_1 - \sqrt{R_1^2 - \frac{c^2}{4}} \quad (7.9)$$

where

$$c = 2r_t \quad (7.10)$$

$$\theta = 2\alpha \quad (7.11)$$

This makes us able to find the centre point of the transition circle relative to our path circle. Recall the circular path having a centre point at (x_{c1}, y_{c1}, z_{c1}) and the straight line having a starting point at (x_{c2}, y_{c1}, z_{c1}) . The transition circle will then be centered around $(x_{c2} + h + r_t, y_{c1} + r_t, z_{c1})$, where h is given from (7.9). Let θ_t be the angle defining the location along the transition circle. The angular position can now be taken as path coordinate for the transition segment.

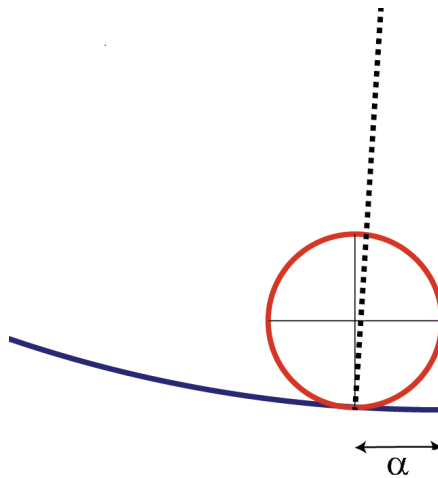


Figure 7.3: Rounding corner between two path segments for path constrained motion. The desired path is shown in blue. The new path will follow the red circle from the point where the dashed line meets the blue curved path segment and continue up the straight line from where the red circle touches the straight line path segment.

Then, we can perform the inverse kinematics calculations for the transition segment path as follows.

$$q_1^*(\theta_t) = \arctan\left(\frac{r_t \sin \theta_t}{r_t \cos \theta_t + x_{c2} + h + r_t}\right) + \arctan\left(\frac{r_t}{x_{c2} + h + r_t}\right) \quad (7.12)$$

$$q_2^*(\theta_t) = \frac{\pi}{2} - \theta_b \quad (7.13)$$

$$q_3^*(\theta_t) = \frac{\pi}{2} - \theta_m \quad (7.14)$$

where

$$\theta_m = \arccos\left(\frac{a_3^2 + a_2^2 - l(\theta_t)}{2a_3a_2}\right) \quad (7.15)$$

$$\theta_b = \frac{a_3 \sin(\theta_m)}{l(\theta_t)} - \theta_f \quad (7.16)$$

$$l(\theta_t) = \sqrt{(d_1 - z_{c1})^2 + (x_{c2} + h + r_t - a_1 + r_t \cos(\theta_t))^2} \quad (7.17)$$

$$\theta_f = \arctan\left(\frac{d_1 - z_{c1}}{x_{c2} + h + r_t - a_1 + r_t \cos(\theta_t)}\right) \quad (7.18)$$

The variables θ_m , θ_b , $l(\theta_t)$ and θ_f are aligned in the same manner as the inverse kinematics of the horizontal circle path from Section 4.3.1. The new path will follow the transition circle from the point where the dashed line meets the blue curved path segment (see Figure 7.3) and continue up the straight line from where the red circle touches the straight line path segment. This means that $\frac{1}{4}$ of the transition circle will be part of the new path, where the transition segment end points are set as $\theta_{t,0} = -\pi$ and $\theta_{t,e} = -\frac{\pi}{2}$. Figure 7.5 shows the evolution of coordinates for

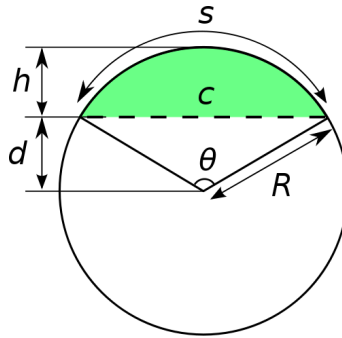


Figure 7.4: Illustrating the geometry of a circular segment. Illustration from Wikipedia.

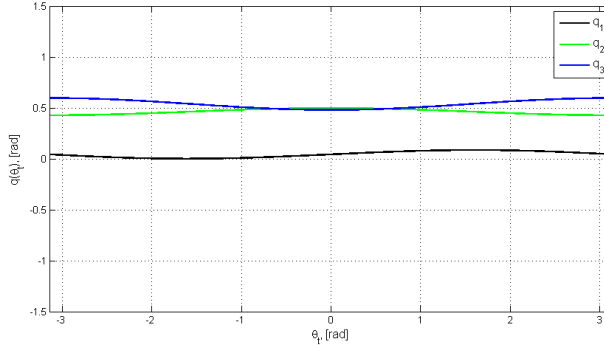


Figure 7.5: Showing planned $q_i^*(\theta_t)$ for the transition segment with radius $r_t = 0.04$ (m). We see that the initial positions align with the corresponding positions of $q_i^*(\theta_1)$ for $\theta_1 = \beta$.

the desired trajectory of the transition segment for $\theta_t \in [-\pi, \pi]$. We can see that initial positions for q_1 , q_2 and q_3 align with the corresponding positions for $q^*(\theta_1)$ for $\theta_1 = \beta$.

7.2.1 Simulations with Transition Segment

The manipulator was simulated for the combined path after implementing the transition segment with $r_t = 0.04$ (m). The velocity profile, $\dot{\theta}_t$, for the transition segment was set to equal initial velocity as the final velocity of the velocity profile, $\dot{\theta}_1$ for the horizontal circle path segment. Recall from Section 6.2.1, that the base joint, q_1 was the only joint failing tracking of its desired path.

Figure 7.6 shows the obtained evolution of the coordinates q_1 , q_2 and q_3 after implementing the transition segment. The area of effect from the transition segment is highlighted for the first joint, q_1 . If we compare this with the same simulation from Section 6.2.1, where the path was simulated without the transition segment (see Figure 6.5), we see that the first joint is now able to track the desired trajectory. The tracking is obviously not perfect due to the nature of the method, where we round the corner to obtain a softer connection point.

Figure 7.7 shows the resulting trajectory in task space plotted with the desired trajectory, excluding the transition segment. We see that we are able to track the desired trajectory, at the cost of a deviation spike at the connection point for $y(t)$. The total period of motion, when using zero initial velocity for the horizontal circle path segment, zero final velocity for the straight line path segment and boundary velocity through the transition segment connection was found to be $T_{ts,f1} = 1.4476$ (s).

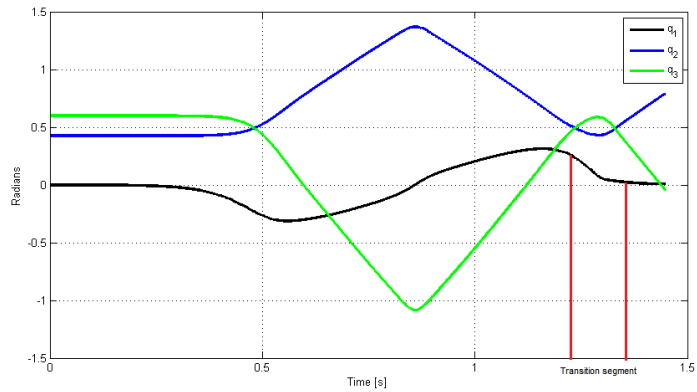


Figure 7.6: Resulting trajectory in joint space as functions of $\theta \in [-\pi, \beta]$ after implementing the transition path segment. Transition segment is using lower velocity profile, resulting in higher path accuracy. Highlighted area (between the red lines) shows the area of effect from the transition segment.

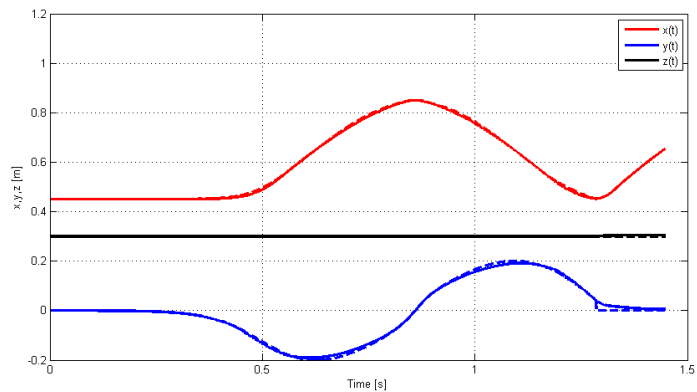


Figure 7.7: Resulting trajectory in task space after implementing the transition path segment.

Compared to the simulated period of motion from Section 6.2.1 with zero velocities at the connection point (1.509 (s)), we can see that the time is improved. It is slightly slower than the period of motion with boundary level connection velocities, without the transition segment implementation, seen in Section 6.2.1. This case, however, failed to track the desired trajectory. Using boundary velocities throughout the entire combined path, we found a period of motion of $T_{ts,f2} = 1.045(\text{s})$, whereas the planned time in this situation can be found as $T_{f,1} + T_{f,3} = 1.044(\text{s})$. We see that the transition segment solution is able to keep up with the planned (sub-)optimal period of motion also in this situation.

7.2.2 Design Options

Varying the radius length, r_t , of the transition circle will obviously impact the results. A larger radius means we can achieve more robust tracking for high velocities during the transition segment. This will however come at the cost of larger deviation from the original desired path. Choosing a small r_t lets us come closer to the original desired path. This also means we need to set a lower velocity profile during the transition segment to be able to obtain robust and accurate tracking. Finding the 'best possible r_t ' may not always be easy. For preplanned paths where we have the possibility to perform simulations, we can simulate with varying r_t (and α), and

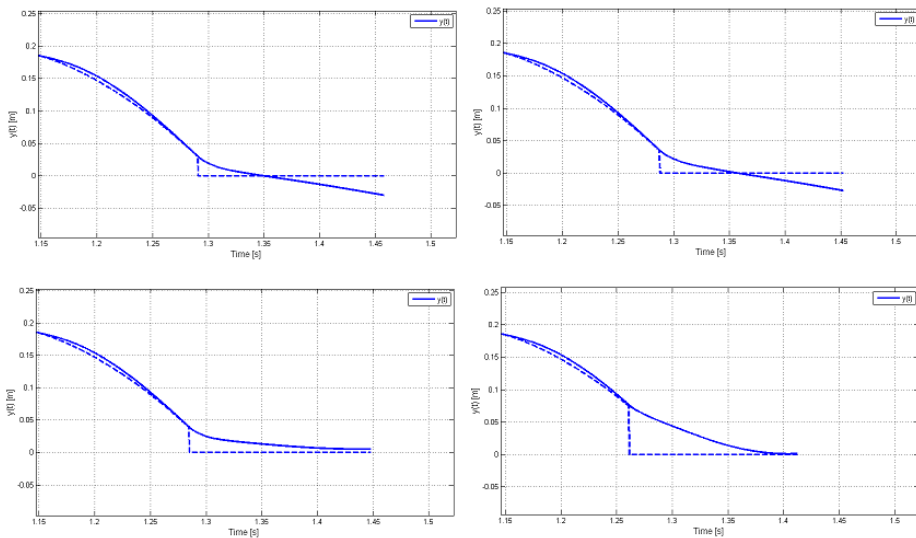


Figure 7.8: Highlighting the effect of tuning the transition segment radius for the obtained $y(t)$ (blue) with desired trajectory (dashed blue). *Top left:* $r_t = 0.03$. *Top right:* $r_t = 0.035$. *Bottom left:* $r_t = 0.04$. *Bottom right:* $r_t = 0.08$. (m)

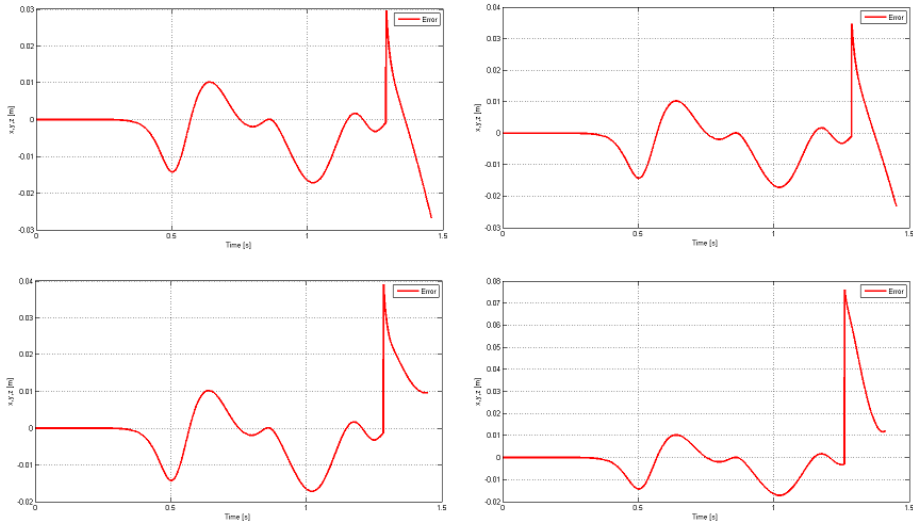


Figure 7.9: Error in task space after implementing the transition path segment, showing the result for four different r_t of the transition segment. *Top left:* $r_t = 0.03$. *Top right:* $r_t = 0.035$. *Bottom left:* $r_t = 0.04$. *Bottom right:* $r_t = 0.08$. (m)

choose the r_t giving the least possible average error combined with the lowest possible time period, over a chosen time span. The case of the horizontal circle connected to a straight line, with a transition segment around the corner point was simulated for several transition circle radiuses, r_t . In Figure 7.7 the resulting trajectory in task space from the simulation in the previous section can be seen. As mentioned earlier, a transition circle radius of $r_t = 0.04$ (m) was used in this case. We see that $y(t)$ (blue) has a deviation from the desired trajectory (dashed) around $t = 1.3$ where the transition segment operates. In Figure 7.8 we can see the results of the simulations in task space, highlighted for $y(t)$, for four different transition circle radiuses. The resulting total error in task space from these simulations can be seen in Figure 7.9. We can observe that the results of using $r_t < 0.4$ results in crossover issues for the tracking of the desired trajectory (dashed line). Using $r_t > 0.4$ did not give this problem, but results in high deviation errors. Choosing $r_t \approx 0.04$ seems to be a good choice in this case. In addition to r_t , the velocity profile of the transition segment will also affect the result. The velocity profile of the transition segment was chosen to reflect the velocity profile for the horizontal circle path for the trajectory results presented in Figure 7.6 and Figure 7.7. In Figure 7.10 we see the effect of tuning the velocity profile of the transition segment, highlighted for $y(t)$. We see that setting too low and too high velocity profile values gives worse results. These results show that spending time tuning the velocity profiles and the transition circle radius r_t

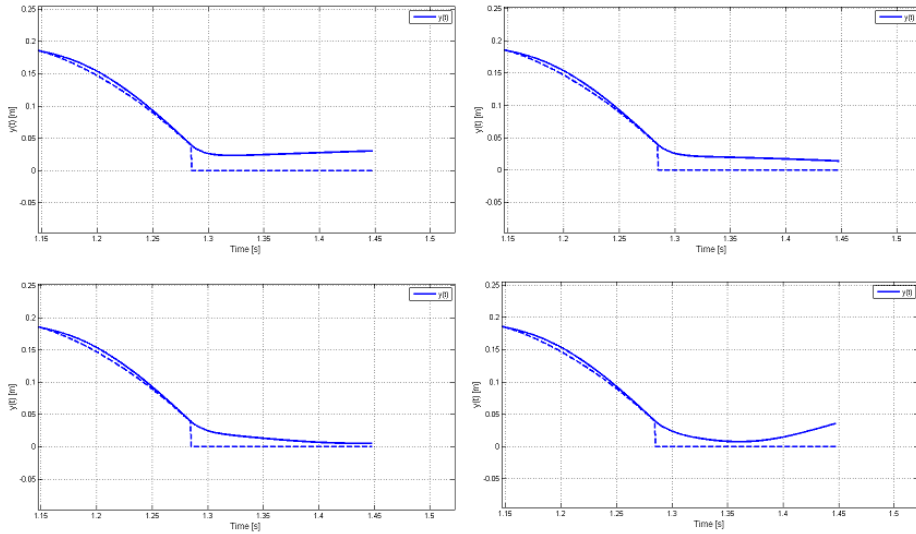


Figure 7.10: Highlighting the effect of tuning the transition segment velocity profile on the obtained $y(t)$ (blue) with desired trajectory (dashed blue). *Top left:* zero velocity profile settings. *Top right:* Low velocity profile settings. *Bottom left:* velocity profile equal to horizontal circle velocity profile. *Bottom right:* higher velocity profile settings.

may give better performance.

The robustness of this method can be an issue if the sampling frequency of the controller causes us to not be able to detect when the path coordinate reaches the value β fast enough. The standard ABB controller has a sampling frequency of $\Delta_s = 0.004(\text{s})$, and simulations in this paper have used sampling frequency equal to that of the ABB controller. A possible way to improve robustness, might be to implement many different transition segments. For the presented case, this would mean implementing many different transition circles with various radius lengths, as illustrated in Figure 7.12, so that the robot will choose at least one of these circles as its path in the transition segment startpoint, and not skip it completely. This will however only be an issue for very small transition circles.

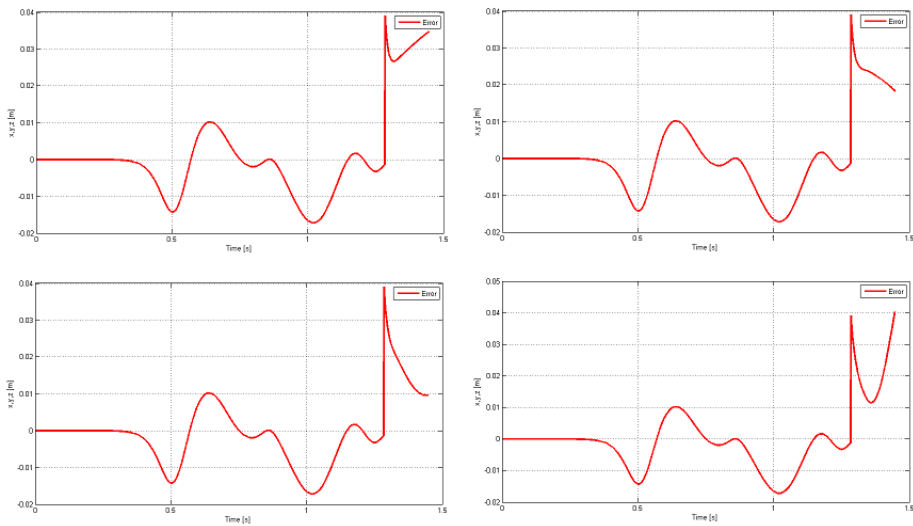


Figure 7.11: Error vector in task space after implementing the transition path segment, showing the result for four different velocity profile settings on the transition segment.

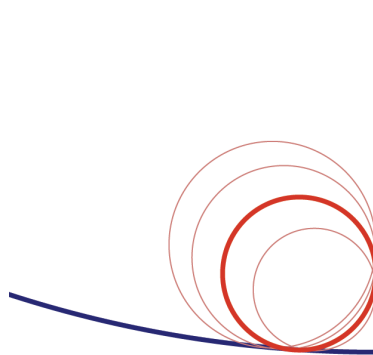


Figure 7.12: A more robust way of implementing, adding more possible transition paths with various level of precision. The circle with a thicker line shows the same circle as in Figure 7.3.

7.3 Discussion

In this chapter we have attempted to find a solution to the problem of sharp corners when connecting path segments for path constrained motion. This problem surfaced from the results in Chapter 6, where we observed that tracking failed for 90 degree corner connections when using boundary level velocities at the connection points. We have seen how implementing a transition segment around the corner point solves the problem of tracking failure. This comes at the cost of some error from the originally desired trajectory. The deviation spike resulting from the method was seen to be ≈ 40 (mm) for a 90 degree corner with boundary level velocities. It might be counterintuitive to use this solution for path constrained motion, because we usually want a desired trajectory to be tracked exactly for such motions. It is, however, argued that the method may be helpful for scenarios where the time optimality criterium is rated higher than exact path tracking accuracy, especially at the particular section of the path where the segments are connected. The results show that the controller achieves high precision for other sections of the paths. Trading for the path deviation error, we get a time improvement of the period of motion, from 1.728 (s) to 1.4476 (s), which is significant.

Chapter 8

Concluding Remarks

In the search for ways to optimize industrial robot manipulator performance, this paper has investigated several important steps needed to achieve robust and effective motion planning and control with regards to time optimization.

The problem of trajectory planning has been investigated using a method called path-constrained trajectory planning. This method parameterizes the nominal evolution of the full state space vector along a path, by using virtual holonomic constraints, without taking the system dynamics into concern. With this approach, the explicit dependence on time disappears, allowing for implementation of a modified control strategy that take advantage of these concepts. For the velocity assignment, various curve fitting tools have been analyzed for generating time (sub-)optimal velocity profiles. The manipulator dynamics were modeled using the Newton-Euler formulation. Using the scenario of a horizontal circular path the method gave a planned period of motion of $T_{s,\theta_1} \approx 0.917$ seconds for the IRB140, which is more than 25% faster than the commercial ABB planner running on maximum velocity. When simulating the planned trajectory with boundary level end point velocity, complemented by a feedback control strategy that achieves orbital stabilization, the period of motion was found to be $T_{f,\theta_1} \approx 0.901$ seconds. Setting the end point velocities to zero resulted in a period of motion of $T_{s,\theta_1} \approx 1.502$ seconds, compared to a planned time of approximately 1.559 seconds. The control structure proved to be an efficient tool for achieving accurate tracking of the preplanned trajectories. The maximum deviation between desired and obtained trajectory was seen as 0.0002 radians for zero velocity end point settings. We have presented the results of similar simulations performed on various path scenarios.

We have investigated the possibilities of connecting path segments, and simulated the manipulator for different cases. When connecting a horizontal circle path to a straight line path up the circle radius, we established a new path with a 90 degree corner point. We observed from simulations that tracking of the desired trajectory

failed for one of the joints using boundary level velocity at the connection point. However, when using zero velocity at the connection point, we were able to track the desired path with only minor deviations. A horizontal circle path was connected to a vertical circle path, which made for a "softer" connection point. In this case we were able to traverse the connection point with boundary level velocity for both velocity profiles. The conclusion that was drawn from the results was that immediate changes in the velocity profile is not possible without using paths that were geometrically conditioned to be connected.

An attempt was made to deal with this problem, using a "transition segment", to smooth out the motion of the corner point. We observed that using the transition segment, we were able to track the desired trajectory with a deviation of 40 mm using boundary level velocity for the two path segments. It was concluded that this solution can be useful, especially if time optimization is regarded as more important than accurate path tracking, e.g. packing tasks. We also showed how to tune the transition segment to obtain more satisfying results. For less sharp angles, this solution should provide better results than for sharp angles such as the 90 degree corner presented in this paper. This would be of interest to investigate in further work. It is likely that more work on the subject can lead to even better solutions, or improvements of the suggested solution presented in this paper. A natural next step would be to test the results on a real life manipulator. Unfortunately, the manipulator was unavailable due to maintenance work for most of the time spent working on this thesis, so the decision was made to focus the experiments on simulations.

Bibliography

- [1] Arturo Baroncelli. Press release: President's report - 2013 set new record in sales. <http://www.ifr.org>, 2014.
- [2] J Bobrow, S Dubowsky, and J Gibson. Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, (3), 1985.
- [3] Jianguing Fan and Qiwei Yao. *Nonlinear Time Series: Nonparametric and Parametric Methods*. Springer-Verlag New York, Inc., 2003.
- [4] ABB Group. Datasheet for IRB 140 Industrial Robot. 2010.
- [5] ABB Group. IRB 140 CAD models. <http://new.abb.com/products/robotics/industrial-robots/irb-140/irb-140-cad>, 2014.
- [6] J Hollerbach. Dynamic scaling of manipulator trajectories. *ASME Journal of Dynamic Systems, Measurement and Control*, (1), 1984.
- [7] T.R. Kurfess. *Robotics and automation handbook*. CRC Press, London., 2005.
- [8] G. Leonov. *Generalization of the Andronov-Vitt theorem*, volume 11. Regular and Chaotic Dynamics, 2006.
- [9] J Lloyd and V Hayward. Trajectory generation for sensor-driven and time-varying tasks. *International journal of robotics research*, (12), 1993.
- [10] Stephen M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.
- [11] S Macfarlane and E Croft. Jerk-bounded manipulator trajectory planning: design for real-time applications. *Transactions on robotics and automation*, (19), 2003.
- [12] Jorge Nocedal and Stephen Wright. *Springer Series in Operations Research: Numerical Optimization*, volume 2. Springer, 2006.
- [13] Marius Nordheim Røv. Time optimal control of industrial manipulators. *Project work, NTNU*, 2013.
- [14] International Federation of Robotics. ISO 8373 - industrial robot definition. <http://www.ifr.org/industrial-robots/>, 2013.

-
- [15] Daniel Ortiz Morales, Simon Westerberg, Pedro La Hera, Uwe Mettin, Leonid B. Freidovich, and Anton S. Shiriaev. Open-loop control experiments on driver assistance for crane forestry machines. *Robotics and Automation (ICRA), IEEE International Conference*, 2011.
- [16] B. Paden and R. Panja. Globally asymptotically stable pd+ controller for robot manipulators. *Int. J. of Contr.*, 47, 1988.
- [17] F Pfeiffer and R Johanni. A concept for manipulator trajectory planning. *IEEE journal of robotics and automation*, (3), 1987.
- [18] Stepan S. Pchelkin, Anton S. Shiriaev, Anders Robertsson, and Leonid B. Freidovich. Integrated time-optimal trajectory planning and control design for industrial robot manipulator. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.
- [19] Anton S. Shiriaev, Leonid B. Freidovich, and Sergei V. Gusev. Transverse linearization for controlled mechanical systems with several passive degrees of freedom. *Automatic Control, IEEE Transactions*, 55(4), 2010.
- [20] Anton S. Shiriaev, John Perram, Anders Robertsson, and Anders Sandberg. Periodic motion planning for virtually constrained euler-lagrange systems. *Systems and Control Letters*, 55:900–907, 2006.
- [21] Kang Shin and N McKay. A dynamic programming approach to trajectory planning of robotic manipulators. *Automatic Control, IEEE Transactions*, (6), 1986.
- [22] M. Takegaki and S. Arimoto. A new feedback method for dynamic control of manipulators. *Journal of Dynamic Syst., Meas., and Control - Trans of ASME*, 103(2), 1981.
- [23] van Dijk N.J.M. Generic trajectory generation for industrial manipulators. *Eindhoven University of Technology, Department of Mechanical Engineering, Dynamics and Control*, 2006.
- [24] D Verscheure, B Demeulenaere, J Swevers, and J De Schutter. Time-optimal path tracking for robots: A convex optimization approach. *Automatic Control, IEEE Transactions*, (10), 2009.
- [25] D. W. Jordan and P Smith. *Nonlinear Ordinary Differential Equations*. Oxford University Press, 2007.
- [26] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. Springer, 2006.

Appendix

Derivations and Definitions

For Matlab simulations the system needs to be transformed into a first-order nonlinear form. The way this was done for the dynamic model of the IRB140 is presented in Appendix A.1.

Appendix A.2 shows the transformation matrix from Section 3.1.1.

In Section 3.1.3 we found the Jacobian for linear velocity, expressed as a 3×3 -matrix. This matrix can be seen in Appendix A.3.

Appendix A.4 shows the inertia matrices from Section 3.2.3, while Section A.5 shows how SolidWorks was used for parameter estimation.

In Section A.6, we see the plots used for motion planning in Chapter 4.

A.1 Order Reduction for Matlab Simulations

Consider the first-order nonlinear form

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

From the dynamic equation on matrix form we get

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

where $M(q)$ is invertible. Now, the system can be reduced from m second-order equations to $2m$ first-order equations by defining

$$\begin{aligned} x_1 &= q_1, & x_2 &= \dot{x}_1 = \dot{q}_1 \\ x_3 &= q_2, & x_4 &= \dot{x}_3 = \dot{q}_2 \\ &\vdots & &\vdots \\ x_{2m-1} &= q_m, & x_{2m} &= \dot{x}_{2m-1} = \dot{q}_m \end{aligned}$$

We can now get the suitable reduced form as follows

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f_2(x, u) \\ \dot{x}_3 &= x_4 \\ \dot{x}_4 &= f_4(x, u) \\ &\vdots \\ \dot{x}_{2m-1} &= x_{2m} \\ \dot{x}_{2m} &= f_{2m}(x, u) \end{aligned}$$

A.2 Transformation Matrix

$$\mathbf{T}_3^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & d_x \\ r_{21} & r_{22} & r_{23} & d_y \\ r_{31} & r_{32} & r_{33} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{A.1})$$

in which

$$\begin{aligned} r_{11} &= -\cos(q_1)[\sin(q_2)\sin(q_3) - \cos(q_2)\cos(q_3)] \\ r_{21} &= -\sin(q_1)[\sin(q_2)\sin(q_3) - \cos(q_2)\cos(q_3)] \\ r_{31} &= -\cos(q_2)\sin(q_3) - \sin(q_2)\cos(q_3) \\ r_{12} &= -\cos(q_1)[\sin(q_2)\cos(q_3) + \cos(q_2)\sin(q_3)] \\ r_{22} &= -\sin(q_1)[\sin(q_2)\cos(q_3) + \cos(q_2)\sin(q_3)] \\ r_{32} &= \sin(q_2)\sin(q_3) - \cos(q_2)\cos(q_3) \\ r_{13} &= -\sin(q_1) \\ r_{23} &= \cos(q_1) \\ r_{33} &= 0 \\ d_x &= -\cos(q_1)[\sin(q_2)a_3\sin(q_3) - a_3\cos(q_2)\cos(q_3) - a_2\sin(q_2) - a_1] \\ d_y &= -\sin(q_1)[a_3\sin(q_2)\sin(q_3) - a_3\cos(q_2)\cos(q_3) - a_2\sin(q_2) - a_1] \\ d_z &= a_2\cos(q_2) + d_1 - a_3\cos(q_2)\sin(q_3) - a_3\sin(q_2)\cos(q_3) \end{aligned}$$

A.3 Jacobian for Linear Velocity

$$\mathbf{J}_{\mathbf{v}3 \times 3} = \begin{bmatrix} \dot{j}_{11} & \dot{j}_{12} & \dot{j}_{13} \\ \dot{j}_{21} & \dot{j}_{22} & \dot{j}_{23} \\ \dot{j}_{31} & \dot{j}_{32} & \dot{j}_{33} \end{bmatrix} \quad (\text{A.2})$$

in which

$$\begin{aligned} \dot{j}_{11} &= \sin(q_1)[a_3 \sin(q_2) \sin(q_3) - a_3 \cos(q_2) \cos(q_3) - a_2 \sin(q_2) - a_1] \\ \dot{j}_{21} &= \cos(q_1)[-a_3 \sin(q_2) \sin(q_3) + a_3 \cos(q_2) \cos(q_3) + a_2 \sin(q_2) + a_1] \\ \dot{j}_{31} &= 0 \\ \dot{j}_{12} &= \cos(q_1)[a_3 \cos(q_2) \sin(q_3) - a_3 \sin(q_2) \cos(q_3) + a_2] \\ \dot{j}_{22} &= \sin(q_1)[-a_3 \cos(q_2) \sin(q_3) - a_3 \sin(q_2) \cos(q_3) + a_2 \cos(q_2)] \\ \dot{j}_{32} &= a_3 \sin(q_2) \sin(q_3) - a_3 \cos(q_2) \cos(q_3) \\ \dot{j}_{13} &= \cos(q_1)[-a_3 \sin(q_2) \cos(q_3) - a_3 \cos(q_2) \sin(q_3)] \\ \dot{j}_{23} &= \sin(q_1)[-a_3 \sin(q_2) \cos(q_3) - a_3 \cos(q_2) \sin(q_3)] \\ \dot{j}_{33} &= a_3 \sin(q_2) \sin(q_3) - a_3 \cos(q_2) \cos(q_3) \end{aligned}$$

A.4 Inertia Matrices

$$I_1 = \begin{bmatrix} 0.51205253974 & 0.00136134 & 0.051305 \\ 0.00136134 & 0.46407468859 & 0.0703356 \\ 0.051305 & -0.0627205 & 0.55411384358 \end{bmatrix} \quad (\text{A.3})$$

$$I_2 = \begin{bmatrix} 0.0948179 & -0.00385971 & 0.037932 \\ -0.00385971 & 0.32860416324 & -0.00108897 \\ 0.037932 & -0.00108897 & 0.27746300488 \end{bmatrix} \quad (\text{A.4})$$

$$I_3 = \begin{bmatrix} 0.50006091595 & -0.00186325 & 0.000934876 \\ -0.00186325 & 0.0751527 & -0.0152041 \\ 0.000934876 & -0.0152041 & 0.51542475434 \end{bmatrix} \quad (\text{A.5})$$

A.5 SolidWorks Parameter Estimation

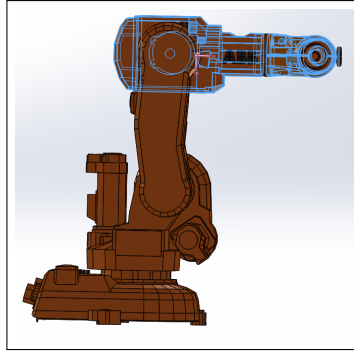


Figure A.1: Showing how the mass center of link 3 was calculated in SolidWorks.

SolidWorks was used for finding link masses and mass centers. We have used CAD models of the IRB140, which can be downloaded from [5]. In Figure A.1 it is shown how the manipulator parameters were obtained for the reduced to 3-DOF model by combining links.

A.6 Velocity Profile Generation Points

Table A.1: Points used in the cubic spline interpolation for the horizontal circle.

Point	1	2	3	4	5	6	7	8	9	10	11	12	13
θ_1	$-\pi$	-2.7	-2.3	-1.8	-1	-0.35	0	0.35	1	1.8	2.3	2.7	π
$\dot{\theta}_1$	$\dot{\theta}_{1,0}$	8.7	8.5	6.6	5.4	7.1	10	7.1	5.4	6.6	8.5	8.7	$\dot{\theta}_{1,e}$

Table A.2: Points used in the cubic spline interpolation for the vertical circle.

Point	1	2	3	4	5	6	7	8	9	10	11
θ_2	$-\pi$	-2.6	-1.8	-1.3	-0.6	0	0.6	1.3	1.8	2.6	π
$\dot{\theta}_2$	$\dot{\theta}_{2,0}$	9	11.5	12.3	9.6	7.7	9.6	12.3	11.5	9	$\dot{\theta}_{2,e}$

Table A.3: Points used in the cubic spline interpolation for the straight line with boundary level velocities at the starting and ending points.

Point	1	2	3
x	0.45	0.55	0.65
\dot{x}	1.55	1.42	1.22

Table A.4: Points used in the cubic spline interpolation for the straight line with zero velocities at the starting and ending points.

Point	1	2	3	4	5	6	7
x	0.45	0.48	0.50	0.55	0.60	0.62	0.65
\dot{x}	0	1.40	1.50	1.40	1.35	1.18	0

Table A.5: Points used in the cubic spline interpolation for the straight line with boundary level velocity at the starting point and zero at the ending point.

Point	1	2	3	4	5
x	0.45	0.55	0.60	0.62	0.65
\dot{x}	1.55	1.42	1.35	1.18	0

Appendix **B**

Maple Code

Maple Code was used for throughout the paper for various calculations. Especially, computing the Newton-Euler formulation leading to the dynamic model of the IRB140, and the computations for the velocity assignment. The code is presented below, and the runnable files can be found in the file folder related to the thesis.

```

[> restart
[> with(LinearAlgebra) :
[> with(CodeGeneration) :
[> with(CurveFitting) :
[> with(plots) :

```

[KINEMATICS

[*Defining known lengths*

```

[> a1 := 0.07 ;; a2 := 0.36 ;; a3 := 0.445 ;; d1 := 0.352 :

```

[*Joint definitions*

```

[> q := Vector([q1(t), q2(t), q3(t)]) :
[> Dq := map(diff, q, t) :
[> DDq := map(diff, Dq, t) :

```

[*Calculating the translation matrix*

```

[> A1 := 
$$\begin{bmatrix} \cos(q[1]) & 0 & -\sin(q[1]) & a1 \cdot \cos(q[1]) \\ \sin(q[1]) & 0 & \cos(q[1]) & a1 \cdot \sin(q[1]) \\ 0 & -1 & 0 & d1 \\ 0 & 0 & 0 & 1 \end{bmatrix} :$$

[> A2 := 
$$\begin{bmatrix} \sin(q[2]) & \cos(q[2]) & 0 & a2 \cdot \sin(q[2]) \\ -\cos(q[2]) & \sin(q[2]) & 0 & -a2 \cdot \cos(q[2]) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} :$$

[> A3 := 
$$\begin{bmatrix} -\sin(q[3]) & -\cos(q[3]) & 0 & -a3 \cdot \sin(q[3]) \\ \cos(q[3]) & -\sin(q[3]) & 0 & a3 \cdot \cos(q[3]) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} :$$

[> T03 := MatrixMatrixMultiply(MatrixMatrixMultiply(A1, A2), A3) :

```


Calculating end effector coordinates

```
> Ident1 :=  $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$  :
```

```
> Ident2 :=  $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$  :
```

```
> Ident3 :=  $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$  :
```

```
> P0 := MatrixMatrixMultiply(MatrixMatrixMultiply(Ident1, T03), Ident2) :
```

```
> #Matlab(P0[1], resultname = "p0x");
```

```
> #Matlab(P0[2], resultname = "p0y");
```

```
> #Matlab(P0[3], resultname = "p0z");
```

Calculating rotation matrices

```
> R01 := MatrixMatrixMultiply(MatrixMatrixMultiply(Ident1, A1), Ident3) :
```

```
> R12 := MatrixMatrixMultiply(MatrixMatrixMultiply(Ident1, A2), Ident3) :
```

```
> R23 := MatrixMatrixMultiply(MatrixMatrixMultiply(Ident1, A3), Ident3) :
```

```
> R02 := MatrixMatrixMultiply(R01, R12) :
```

```
> R03 := MatrixMatrixMultiply(R02, R23) :
```

Calculating rotation axis of each joint expressed in its own frame

```
>
```

```
> z0 :=  $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$  :
```

```
> b1 := MatrixMatrixMultiply(Transpose(R01), z0) :
```

```
> b2 := combine(MatrixMatrixMultiply(Transpose(R02), MatrixVectorMultiply(R01, z0)),  
trig) :
```

```
> b3 := combine(MatrixMatrixMultiply(Transpose(R03), MatrixVectorMultiply(R02, z0)),  
trig) :
```

[DYNAMICS

[*Defining the initial gravity vector*

$$\begin{aligned} > \text{g0} := \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} : \end{aligned}$$

[*Defining the link masses (in kg) with values from SolidWorks*

$$\begin{aligned} > \text{m1} &:= 34.655 : \\ > \text{m2} &:= 15.994 : \\ > \text{m3} &:= 20.862 : \end{aligned}$$

[*Defining the Inertia Matrices with values from SolidWorks*

$$\begin{aligned} > \\ > \text{I1} &:= \begin{bmatrix} 0.51205253974 & -0.00385971 & 0.051305 \\ 0.00136134 & 0.46407468859 & 0.0703356 \\ 0.051305 & -0.0627205 & 0.55411384358 \end{bmatrix} : \\ > \text{I2} &:= \begin{bmatrix} 0.0948179 & -0.00385971 & 0.037932 \\ -0.00385971 & 0.32860416324 & -0.00108897 \\ 0.037932 & -0.00108897 & 0.27746300488 \end{bmatrix} : \\ > \text{I3} &:= \begin{bmatrix} 0.50006091595 & -0.00186325 & 0.000934876 \\ -0.00186325 & 0.0751527 & -0.0152041 \\ 0.000934876 & -0.0152041 & 0.51542475434 \end{bmatrix} : \end{aligned}$$

[*Defining mass center vectors with values from SolidWorks*

$$\begin{aligned} > \text{r0c1} &:= \text{Vector}([[0.08903], [-0.02789], [0.04312]]) : \\ > \text{r1c2} &:= \text{Vector}([[0.19829], [-0.09243], [0.00973]]) : \\ > \text{r2c3} &:= \text{Vector}([[0.07996], [0.00456], [0.00586]]) : \\ > \text{r01} &:= \text{Vector}([[a1], [-d1], [0]]) : \\ > \text{r12} &:= \text{Vector}([[a2], [0], [0]]) : \\ > \text{r23} &:= \text{Vector}([[a3], [0], [0]]) : \\ > \text{r1c1} &:= \text{r0c1} - \text{r01} : \\ > \text{r2c2} &:= \text{r1c2} - \text{r12} : \\ > \text{r3c3} &:= \text{r2c3} - \text{r23} : \end{aligned}$$

[*Newton – Euler recursions*

[*Forward Recursion for Link 1:*

- > $\omega_1 := b_1 \cdot Dq[1] :$
- > $\alpha_1 := b_1 \cdot DDq[1] + \text{CrossProduct}(\omega_1, b_1 \cdot Dq[1]) :$
- > $D\omega_1 := \text{map}(\text{diff}, \omega_1, t) :$
- > $ae_1 := \text{CrossProduct}(D\omega_1, r_{01}) + \text{CrossProduct}(\omega_1, \text{CrossProduct}(\omega_1, r_{01})) :$
- > $ac_1 := \text{CrossProduct}(D\omega_1, r_{0c1}) + \text{CrossProduct}(\omega_1, \text{CrossProduct}(\omega_1, r_{0c1})) :$
- > $g_1 := \text{MatrixMatrixMultiply}(\text{Transpose}(R_{01}), g_0) :$

[*Forward Recursion for Link 2:*

- > $\omega_2 := \text{combine}(\text{MatrixVectorMultiply}(\text{Transpose}(R_{12}), \omega_1) + b_2 \cdot Dq[2], \text{trig}) :$
- > $\alpha_2 := \text{combine}(\text{MatrixVectorMultiply}(\text{Transpose}(R_{12}), \alpha_1) + b_2 \cdot DDq[2] + \text{CrossProduct}(\omega_2, b_2 \cdot Dq[2]), \text{trig}) :$
- > $D\omega_2 := \text{map}(\text{diff}, \omega_2, t) :$
- > $ae_2 := \text{combine}(\text{MatrixVectorMultiply}(\text{Transpose}(R_{12}), ae_1) + \text{CrossProduct}(D\omega_2, r_{12}) + \text{CrossProduct}(\omega_2, \text{CrossProduct}(\omega_2, r_{12})), \text{trig}) :$
- > $ac_2 := \text{combine}(\text{MatrixVectorMultiply}(\text{Transpose}(R_{12}), ac_1) + \text{CrossProduct}(D\omega_2, r_{1c2}) + \text{CrossProduct}(\omega_2, \text{CrossProduct}(\omega_2, r_{1c2})), \text{trig}) :$
- > $g_2 := \text{combine}(\text{MatrixVectorMultiply}(\text{Transpose}(R_{02}), g_0), \text{trig}) :$

[*Forward Recursion for Link 3:*

- > $\omega_3 := \text{combine}(\text{MatrixVectorMultiply}(\text{Transpose}(R_{23}), \omega_2) + b_3 \cdot Dq[3], \text{trig}) :$
- > $\alpha_3 := \text{combine}(\text{MatrixVectorMultiply}(\text{Transpose}(R_{23}), \alpha_2) + b_3 \cdot DDq[3] + \text{CrossProduct}(\omega_3, b_3 \cdot Dq[3]), \text{trig}) :$
- > $D\omega_3 := \text{map}(\text{diff}, \omega_3, t) :$
- > $ae_3 := \text{combine}(\text{MatrixVectorMultiply}(\text{Transpose}(R_{23}), ae_2) + \text{CrossProduct}(D\omega_3, r_{23}) + \text{CrossProduct}(\omega_3, \text{CrossProduct}(\omega_3, r_{23})), \text{trig}) :$
- > $ac_3 := \text{combine}(\text{MatrixVectorMultiply}(\text{Transpose}(R_{23}), ac_2) + \text{CrossProduct}(D\omega_3, r_{2c3}) + \text{CrossProduct}(\omega_3, \text{CrossProduct}(\omega_3, r_{2c3})), \text{trig}) :$
- > $g_3 := \text{combine}(\text{MatrixVectorMultiply}(\text{Transpose}(R_{03}), g_0), \text{trig}) :$

[*Backward Recursion for Link 3*

- > $f_3 := \text{Add}(m_3 \cdot ac_3, -m_3 \cdot g_3) :$
- > $\tau_3 := -\text{CrossProduct}(f_3, r_{2c3}) + \text{MatrixVectorMultiply}(I_3, \alpha_3) + \text{CrossProduct}(\omega_3, \text{MatrixVectorMultiply}(I_3, \omega_3)) :$
- > $\tau_{3z} := \text{collect}(\text{combine}(\text{MatrixVectorMultiply}(\text{Transpose}(b_3), \tau_3), \text{trig}), \{\text{DDq}[1], \text{DDq}[2], \text{DDq}[3], \text{Dq}[1], \text{Dq}[2], \text{Dq}[3]\}) :$

Backward Recursion for Link 2

```
> f2 := MatrixVectorMultiply(R23, f3) + Add(m2·ac2, -m2·g2) :  
> τ2 := MatrixVectorMultiply(R23, τ3) - CrossProduct(f2, r1c2)  
      + CrossProduct(MatrixVectorMultiply(R23, f3), r2c2) + MatrixVectorMultiply(I2, α2)  
      + CrossProduct(ω2, MatrixVectorMultiply(I2, ω2)) :  
> τ2z := collect(combine(MatrixVectorMultiply(Transpose(b2), τ2), trig), {DDq[1], DDq[2],  
      DDq[3], Dq[1], Dq[2], Dq[3]}) :
```

Backward Recursion for Link 1

```
> f1 := MatrixVectorMultiply(R12, f2) + Add(m1·ac1, -m1·g1) :  
> τ1 := MatrixVectorMultiply(R12, τ2) - CrossProduct(f1, r0c1)  
      + CrossProduct(MatrixVectorMultiply(R12, f2), r1c1) + MatrixVectorMultiply(I1, α1)  
      + CrossProduct(ω1, MatrixVectorMultiply(I1, ω1)) :  
> τ1z := collect(combine(MatrixVectorMultiply(Transpose(b1), τ1), trig), {DDq[1], DDq[2],  
      DDq[3], Dq[1], Dq[2], Dq[3]}) :
```

Defining dynamic system matrix elements

Inertia-matrix

```
> m11 := eval(τ1z, {DDq[1]=1, DDq[2]=0, DDq[3]=0, Dq[1]=0, Dq[2]=0, Dq[3]=0, g  
      =0}) :  
> m12 := eval(τ1z, {DDq[1]=0, DDq[2]=1, DDq[3]=0, Dq[1]=0, Dq[2]=0, Dq[3]=0, g  
      =0}) :  
> m13 := eval(τ1z, {DDq[1]=0, DDq[2]=0, DDq[3]=1, Dq[1]=0, Dq[2]=0, Dq[3]=0, g  
      =0}) :  
> m21 := eval(τ2z, {DDq[1]=1, DDq[2]=0, DDq[3]=0, Dq[1]=0, Dq[2]=0, Dq[3]=0, g  
      =0}) :  
> m22 := eval(τ2z, {DDq[1]=0, DDq[2]=1, DDq[3]=0, Dq[1]=0, Dq[2]=0, Dq[3]=0, g  
      =0}) :  
> m23 := eval(τ2z, {DDq[1]=0, DDq[2]=0, DDq[3]=1, Dq[1]=0, Dq[2]=0, Dq[3]=0, g  
      =0}) :  
> m31 := eval(τ3z, {DDq[1]=1, DDq[2]=0, DDq[3]=0, Dq[1]=0, Dq[2]=0, Dq[3]=0, g  
      =0}) :  
> m32 := eval(τ3z, {DDq[1]=0, DDq[2]=1, DDq[3]=0, Dq[1]=0, Dq[2]=0, Dq[3]=0, g  
      =0}) :  
> m33 := eval(τ3z, {DDq[1]=0, DDq[2]=0, DDq[3]=1, Dq[1]=0, Dq[2]=0, Dq[3]=0, g  
      =0}) :
```

Gravity vector

```
> g1 := eval(τ1z, {DDq[1]=0, DDq[2]=0, DDq[3]=0, Dq[1]=0, Dq[2]=0, Dq[3]=0}) :  
> g2 := eval(τ2z, {DDq[1]=0, DDq[2]=0, DDq[3]=0, Dq[1]=0, Dq[2]=0, Dq[3]=0}) :
```

```
[> g3 := eval(τ3z, {DDq[1]=0, DDq[2]=0, DDq[3]=0, Dq[1]=0, Dq[2]=0, Dq[3]=0}) :
```

```
[> cM1 := eval(τ1z, {DDq[1]=0, DDq[2]=0, DDq[3]=0, g=0}) :
```

```
[> cM2 := eval(τ2z, {DDq[1]=0, DDq[2]=0, DDq[3]=0, g=0}) :
```

```
[> cM3 := eval(τ3z, {DDq[1]=0, DDq[2]=0, DDq[3]=0, g=0}) :
```

```
[> #Matlab(cM1, resultname="c1");
```

```
[> #Matlab(cM2, resultname="c2");
```

```
[> #Matlab(cM3, resultname="c3");
```

[Setting up the dynamic system on matrix form

```
[> M0 := Matrix([ [m11, m12, m13], [m21, m22, m23], [m31, m32, m33] ]) :
```

```
[> G0 := Vector([ [g1], [g2], [g3] ]) :
```

```
[> τ0 := Vector([ [τ1z], [τ2z], [τ3z] ]) :
```

```
[> C0Dq := combine(τ0 - MatrixVectorMultiply(M0, DDq) - G0, trig) :
```

[Setting up the expression for kinetic energy

```
[> Ek :=  $\frac{1}{2}$  · VectorMatrixMultiply(Transpose(Dq), MatrixVectorMultiply(M0, Dq)) :
```

[Code to convert matrix elements to matlab code

```
[> #Matlab(m11, resultname="m11");
```

```
[> #Matlab(m12, resultname="m12");
```

```
[> #Matlab(m13, resultname="m13");
```

```
[> #Matlab(m21, resultname="m21");
```

```
[> #Matlab(m22, resultname="m22");
```

```
[> #Matlab(m23, resultname="m23");
```

```
[> #Matlab(m31, resultname="m31");
```

```
[> #Matlab(m32, resultname="m32");
```

```
[> #Matlab(m33, resultname="m33");
```

```
[> #Matlab(g1, resultname="g1");
```

```
[> #Matlab(g2, resultname="g2");
```

```
[> #Matlab(g3, resultname="g3");
```

```
[> #Matlab(C0Dq(1), resultname="cdq1");
```

```
[> #Matlab(C0Dq(2), resultname="cdq2");
```

```
[> #Matlab(C0Dq(3), resultname="cdq3");
```

[PATH AND TRAJECTORY PLANNING

[*Defining virtual holonomic constraints*

[*Horizontal circular motion*

$$\begin{aligned} > \Phi := \text{Vector}([[\phi_1(\theta_1)], [\phi_2(\theta_1)], [\phi_3(\theta_1)]]): \end{aligned}$$

$$\begin{aligned} > D\Phi := \text{map}(\text{diff}, \Phi, \theta_1): \end{aligned}$$

$$\begin{aligned} > DD\Phi := \text{map}(\text{diff}, D\Phi, \theta_1): \end{aligned}$$

[*Vertical circular motion*

$$\begin{aligned} > \sigma := \text{Vector}([[\sigma_1(\theta_2)], [\sigma_2(\theta_2)], [\sigma_3(\theta_2)]]): \end{aligned}$$

$$\begin{aligned} > D\sigma := \text{map}(\text{diff}, \sigma, \theta_2): \end{aligned}$$

$$\begin{aligned} > DD\sigma := \text{map}(\text{diff}, D\sigma, \theta_2): \end{aligned}$$

[*Straight line motion*

$$\begin{aligned} > \epsilon := \text{Vector}([[\epsilon_1(xsl)], [\epsilon_2(xsl)], [\epsilon_3(xsl)]]): \end{aligned}$$

$$\begin{aligned} > D\epsilon := \text{map}(\text{diff}, \epsilon, xsl): \end{aligned}$$

$$\begin{aligned} > DD\epsilon := \text{map}(\text{diff}, D\epsilon, xsl): \end{aligned}$$

[*Transition segment*

$$\begin{aligned} > \lambda := \text{Vector}([[\lambda_1(\theta_t)], [\lambda_2(\theta_t)], [\lambda_3(\theta_t)]]): \end{aligned}$$

$$\begin{aligned} > D\lambda := \text{map}(\text{diff}, \lambda, \theta_t): \end{aligned}$$

$$\begin{aligned} > DD\lambda := \text{map}(\text{diff}, D\lambda, \theta_t): \end{aligned}$$

[*Inverse Kinematic Calculations*

[*Horizontal circular motion*

$$\begin{aligned} > xc1 := 0.65 ;; yc1 := 0 ;; zc1 := 0.3 ;; R1 := 0.2: \end{aligned}$$

$$\begin{aligned} > l1 := \sqrt{(d1 - zc1)^2 + (xc1 - a1 + R1 \cdot \cos(\theta_1))^2}: \end{aligned}$$

$$\begin{aligned} > \theta_m := \arccos\left(\frac{a2^2 + a3^2 - l1^2}{2 \cdot a2 \cdot a3}\right): \end{aligned}$$

$$\begin{aligned} > \theta_f := \arctan\left(\frac{(d1 - zc1)}{xc1 - a1 + R1 \cdot \cos(\theta_1)}\right): \end{aligned}$$

$$\begin{aligned} > \theta_b := \arcsin\left(\frac{a3 \cdot \sin(\theta_m)}{l1}\right) - \theta_f: \end{aligned}$$

$$\begin{aligned} > \phi_1 := \text{evalf}(\arctan(R1 \cdot \sin(\theta_1), xc1 + R1 \cdot \cos(\theta_1))): \end{aligned}$$

$$\begin{aligned} > \phi_2 := \text{evalf}\left(\frac{\pi}{2} - \theta_b\right): \end{aligned}$$

$$\left[\begin{array}{l} > \phi_3 := \text{evalf}\left(\frac{\pi}{2} - \theta_m\right) : \end{array} \right.$$

Vertical circular motion

$$\left[\begin{array}{l} > xc_2 := 0.45 ;; yc_2 := 0 ;; zc_2 := 0.5 ;; R_2 := 0.2 : \end{array} \right.$$

$$\left[\begin{array}{l} > l_2 := \sqrt{(xc_2)^2 + (R_2 \cdot \sin(\theta_2))^2} - a_1 : \end{array} \right.$$

$$\left[\begin{array}{l} > h := R_2 \cdot \cos(\theta_2) + zc_2 - d_1 : \end{array} \right.$$

$$\left[\begin{array}{l} > \theta_{m2} := \arccos\left(\frac{(a_2^2 + a_3^2 - l_2^2 - h^2)}{2 \cdot a_2 \cdot a_3}\right) : \end{array} \right.$$

$$\left[\begin{array}{l} > \theta_{b2} := \arctan\left(\frac{h}{l_2}\right) : \end{array} \right.$$

$$\left[\begin{array}{l} > \theta_{f2} := \arcsin\left(\frac{a_3 \cdot \sin(\theta_{m2})}{\sqrt{l_2^2 + h^2}}\right) : \end{array} \right.$$

$$\left[\begin{array}{l} > \sigma_1 := \text{evalf}(\arctan(R_2 \cdot \sin(\theta_2), xc_2)) : \end{array} \right.$$

$$\left[\begin{array}{l} > \sigma_2 := \text{evalf}\left(\frac{\pi}{2} - (\theta_{b2} + \theta_{f2})\right) : \end{array} \right.$$

$$\left[\begin{array}{l} > \sigma_3 := \text{evalf}\left(\frac{\pi}{2} - \theta_{m2}\right) : \end{array} \right.$$

Straight line motion

$$\left[\begin{array}{l} > l_3 := \sqrt{(d_1 - zc_1)^2 + (x - a_1)^2} : \end{array} \right.$$

$$\left[\begin{array}{l} > \theta_{m3} := \arccos\left(\frac{(a_2^2 + a_3^2 - l_3^2)}{2 \cdot a_2 \cdot a_3}\right) : \end{array} \right.$$

$$\left[\begin{array}{l} > \theta_{f3} := \arctan\left(\frac{(d_1 - zc_1)}{x - a_1}\right) : \end{array} \right.$$

$$\left[\begin{array}{l} > \theta_{b3} := \arcsin\left(\frac{a_3 \cdot \sin(\theta_{m3})}{l_3}\right) - \theta_{f3} : \end{array} \right.$$

$$\left[\begin{array}{l} > \epsilon_1 := \text{evalf}(0) : \end{array} \right.$$

$$\left[\begin{array}{l} > \epsilon_2 := \text{evalf}\left(\frac{\pi}{2} - \theta_{b3}\right) : \end{array} \right.$$

$$\left[\begin{array}{l} > \epsilon_3 := \text{evalf}\left(\frac{\pi}{2} - \theta_{m3}\right) : \end{array} \right.$$

Transition segment

$$\left[\begin{array}{l} > r_t := 0.02 ;; c := 2 \cdot r_t ;; \beta := 2 \cdot \alpha ;; \end{array} \right.$$

$$\left[\begin{array}{l} > h_t := R_1 - \sqrt{R_1^2 - \frac{c^2}{4}} : \end{array} \right.$$

$$\left[\begin{array}{l} > l_t := \sqrt{(d_1 - zc_1)^2 + (xc_2 + h_t + r_t - a_1 + r_t \cdot \cos(\theta_t))^2} : \end{array} \right.$$

```

[>  $\theta_{m4} := \arccos\left(\frac{a_2^2 + a_3^2 - l_t^2}{2 \cdot a_2 \cdot a_3}\right) :$ 
[>  $\theta_{f4} := \arctan\left(\frac{d_1 - z_{c1}}{x_{c2} + h_{ts} + r_t - a_1 + r_t \cdot \cos(\theta_t)}\right) :$ 
[>  $\theta_{b4} := \arcsin\left(\frac{a_3 \cdot \sin(\theta_{m4})}{l_t}\right) - \theta_{f4} :$ 
[>
[>  $\lambda_1 := \text{evalf}(\arctan(r_t \cdot \sin(\theta_t), x_{c2} + h_{ts} + r_t + r_t \cdot \cos(\theta_t)) + \arctan(r_t, x_{c2} + h_{ts} + r_t)) :$ 
[>  $\lambda_2 := \text{evalf}\left(\frac{\pi}{2} - \theta_{b4}\right) :$ 
[>  $\lambda_3 := \text{evalf}\left(\frac{\pi}{2} - \theta_{m4}\right) :$ 

```

[Plotting the curves for the target trajectories

```

[> hc1 := plot( $\phi_1(\theta_1)$ ,  $\theta_1 = -3.14 .. 3.14$ , color = "Black") :
[> hc2 := plot( $\phi_2(\theta_1)$ ,  $\theta_1 = -3.14 .. 3.14$ , color = "Green") :
[> hc3 := plot( $\phi_3(\theta_1)$ ,  $\theta_1 = -3.14 .. 3.14$ , color = "Blue") :
[> vc1 := plot( $\sigma_1(\theta_2)$ ,  $\theta_2 = -3.14 .. 3.14$ , color = "Black") :
[> vc2 := plot( $\sigma_2(\theta_2)$ ,  $\theta_2 = -3.14 .. 3.14$ , color = "Green") :
[> vc3 := plot( $\sigma_3(\theta_2)$ ,  $\theta_2 = -3.14 .. 3.14$ , color = "Blue") :
[> sl1 := plot( $\epsilon_1(x)$ ,  $x = 0.45 .. 0.65$ , color = "Black") :
[> sl2 := plot( $\epsilon_2(x)$ ,  $x = 0.45 .. 0.65$ , color = "Blue") :
[> sl3 := plot( $\epsilon_3(x)$ ,  $x = 0.45 .. 0.65$ , color = "Green") :
[> ts1 := plot( $\lambda_1(\theta_t)$ ,  $\theta_t = -3.14 .. 3.14$ , color = "Black") :
[> ts2 := plot( $\lambda_2(\theta_t)$ ,  $\theta_t = -3.14 .. 3.14$ , color = "Green") :
[> ts3 := plot( $\lambda_3(\theta_t)$ ,  $\theta_t = -3.14 .. 3.14$ , color = "Blue") :

[> display({hc1, hc2, hc3}) :
[> display({vc1, vc2, vc3}) :
[> display({sl1, sl2, sl3}) :
[> display({ts1, ts2, ts3}) :

```

[Generating upper boundary curves

[Horizontal circle:

```

[> D $\phi_1 := \text{diff}(\phi_1, \theta_1) :$ 
[> D $\phi_2 := \text{diff}(\phi_2, \theta_1) :$ 
[> D $\phi_3 := \text{diff}(\phi_3, \theta_1) :$ 

[> DD $\phi_1 := \text{diff}(D\phi_1, \theta_1) :$ 

```


[> DD ϕ 2 := diff(D ϕ 2, θ 1) :
[> DD ϕ 3 := diff(D ϕ 3, θ 1) :

[> absD ϕ 1 := abs(D ϕ 1) :
[> absD ϕ 2 := abs(D ϕ 2) :
[> absD ϕ 3 := abs(D ϕ 3) :

[> s1hc := $\left(\frac{3.4907}{\text{absD}\phi 1} \right)$:
[> s2hc := $\left(\frac{3.4907}{\text{absD}\phi 2} \right)$:
[> s3hc := $\left(\frac{4.5379}{\text{absD}\phi 3} \right)$:

[*Vertical circle:*

[> D σ 1 := diff(σ 1, θ 2) :
[> D σ 2 := diff(σ 2, θ 2) :
[> D σ 3 := diff(σ 3, θ 2) :

[> DD σ 1 := diff(D σ 1, θ 2) :
[> DD σ 2 := diff(D σ 2, θ 2) :
[> DD σ 3 := diff(D σ 3, θ 2) :

[> absD σ 1 := abs(D σ 1) :
[> absD σ 2 := abs(D σ 2) :
[> absD σ 3 := abs(D σ 3) :

[> s1vc := $\left(\frac{3.4907}{\text{absD}\sigma 1} \right)$:
[> s2vc := $\left(\frac{3.4907}{\text{absD}\sigma 2} \right)$:
[> s3vc := $\left(\frac{4.5379}{\text{absD}\sigma 3} \right)$:

[*Straight line:*

[> D ϵ 1 := diff(ϵ 1, x) :
[> D ϵ 2 := diff(ϵ 2, x) :
[> D ϵ 3 := diff(ϵ 3, x) :

[> DD ϵ 1 := diff(D ϵ 1, x) :
[> DD ϵ 2 := diff(D ϵ 2, x) :
[> DD ϵ 3 := diff(D ϵ 3, x) :

[> absD ϵ 1 := abs(D ϵ 1) :

```
[> absDe2 := abs(De2) :
[> absDe3 := abs(De3) :
```

```
[> s2sl := ( 3.4907 / absDe2 ) :
[> s3sl := ( 4.5379 / absDe3 ) :
```

[Transition segment:

```
[> Dλ1 := diff(λ1, θt) :
[> Dλ2 := diff(λ2, θt) :
[> Dλ3 := diff(λ3, θt) :
```

```
[> DDλ1 := diff(Dλ1, θt) :
[> DDλ2 := diff(Dλ2, θt) :
[> DDλ3 := diff(Dλ3, θt) :
```

```
[> absDλ1 := abs(Dλ1) :
[> absDλ2 := abs(Dλ2) :
[> absDλ3 := abs(Dλ3) :
```

```
[> s1ts := ( 3.4907 / absDλ1 ) :
[> s2ts := ( 3.4907 / absDλ2 ) :
[> s3ts := ( 4.5379 / absDλ3 ) :
```

[Plotting boundary curves used to specify interpolation points

```
[> HCboundary1 := plot(s1hc(θ1), θ1 = -π..π, 3..12, color = "Black") :
[> HCboundary2 := plot(s2hc(θ1), θ1 = -π..π, 3..12, color = "Green") :
[> HCboundary3 := plot(s3hc(θ1), θ1 = -π..π, 3..12, color = "Blue") :
[> VCboundary1 := plot(s1vc(θ2), θ2 = -π..π, 3..16, color = "Black") :
[> VCboundary2 := plot(s2vc(θ2), θ2 = -π..π, 3..16, color = "Green") :
[> VCboundary3 := plot(s3vc(θ2), θ2 = -π..π, 3..16, color = "Blue") :
[> SLboundary2 := plot(s2sl(x), x = 0.45..0.65, 0..3, color = "Green") :
[> SLboundary3 := plot(s3sl(x), x = 0.45..0.65, 0..3, color = "Blue") :
[> TSboundary1 := plot(s1ts(θt), θt = -π..π, 3..100, color = "Black") :
[> TSboundary2 := plot(s2ts(θt), θt = -π..π, 3..100, color = "Green") :
[> TSboundary3 := plot(s3ts(θt), θt = -π..π, 3..100, color = "Blue") :
```

```

[> display( {HCboundary1, HCboundary2, HCboundary3} ) :
[> display( {VCboundary1, VCboundary2, VCboundary3} ) :
[> display( {SLboundary2, SLboundary3} ) :
[> display( {TSboundary1, TSboundary2, TSboundary3} ) :
[>

```

Defining points and generating interpolating curve

(change the values of the points to change velocity profiles)

Horizontal circle:

```

[> pointsHC := [[-3.14159, 7.6], [-2.7, 8.7], [-2.3, 8.5], [-1.8, 6.6], [-1, 5.4], [-0.35, 7.1],
[0, 10], [0.35, 7.1], [1, 5.4], [1.8, 6.6], [2.3, 8.5], [2.7, 8.7], [3.14159, 7.6]] :
[> InterpolyHC := CurveFitting[Spline](pointsHC,  $\theta_1$ , endpoints='natural') :

```

```

[> HCbound1 := plot(s1hc( $\theta_1$ ),  $\theta_1 = -\pi..pi$ , 3..12, color="DarkCyan") :
[> HCbound2 := plot(s2hc( $\theta_1$ ),  $\theta_1 = -\pi..pi$ , 3..12, color="DarkCyan") :
[> HCbound3 := plot(s3hc( $\theta_1$ ),  $\theta_1 = -\pi..pi$ , 3..12, color="DarkCyan") :

```

Vertical circle:

```

[> pointsVC := [[-3.14159, 7.55], [-2.6, 9], [-1.8, 11.5], [-1.3, 12.3], [-0.6, 9.6], [0, 7.7],
[0.6, 9.6], [1.3, 12.3], [1.8, 11.5], [2.6, 9], [3.14159, 7.55]] :
[> InterpolyVC := CurveFitting[Spline](pointsVC,  $\theta_2$ , endpoints='natural') :

```

```

[> VCbound1 := plot(s1vc( $\theta_2$ ),  $\theta_2 = -\pi..pi$ , 3..12, color="DarkCyan") :
[> VCbound2 := plot(s2vc( $\theta_2$ ),  $\theta_2 = -\pi..pi$ , 3..12, color="DarkCyan") :
[> VCbound3 := plot(s3vc( $\theta_2$ ),  $\theta_2 = -\pi..pi$ , 3..12, color="DarkCyan") :

```

Straight line:

```

[> pointsSLv := [[0.45, 0.001], [0.48, 1.4], [0.5, 1.5], [0.65, 1.22]] :
[> pointsSLf := [[0.45, 1.55], [0.55, 1.42], [0.65, 1.22]] :
[> pointsSLs := [[0.45, 0.001], [0.48, 1.4], [0.5, 1.5], [0.55, 1.4], [0.6, 1.35], [0.62, 1.18],
[0.65, 0.001]] :
[> pointsSLr := [[0.45, 1.55], [0.55, 1.42], [0.6, 1.35], [0.62, 1.18], [0.65, 0.001]] :
[> InterpolySL := CurveFitting[Spline](pointsSLf, x, endpoints='natural') :
[> SLbound2 := plot(s2sl(x), x = 0.45..0.65, 0..3, color="DarkCyan") :
[> SLbound3 := plot(s3sl(x), x = 0.45..0.65, 0..3, color="DarkCyan") :

```

Transition segment:

```

[> pointsTS := [[-3.14, 0], [-2.7, 8.7], [-2.3, 8.5], [-1.8, 6.6], [-1, 5.4], [-0.35, 7.1], [0, 10],
[0.35, 7.1], [1, 5.4], [1.8, 6.6], [2.3, 8.5], [2.7, 8.7], [3.14, 0]] :
[> InterpolyTS := CurveFitting[Spline](pointsTS,  $\theta_t$ , endpoints='natural') :
[> TSbound1 := plot(s1ts( $\theta_t$ ),  $\theta_t = -\pi..pi$ , 3..12, color="DarkCyan") :

```

```

[> TSbound2 := plot(s2ts(θt), θt = -π..π, 3..12, color = "DarkCyan") :
[> TSbound3 := plot(s3ts(θt), θt = -π..π, 3..12, color = "DarkCyan") :

```

[*Plotting velocity profiles with boundary curves*]

```

[> HCSplineplot := plot(InterpolyHC, θ1 = -3.14..3.14, 3..12, color = "DarkRed") :
[> VCSplineplot := plot(InterpolyVC, θ2 = -3.14..3.14, 7..17, color = "DarkRed") :
[> SLSplineplot := plot(InterpolySL, x = 0.45..0.65, 0..3, color = "DarkRed") :
[> TSSplineplot := plot(InterpolyTS, θt = -3.14..3.14, 3..12, color = "DarkRed") :

[> display( {HCbound1, HCbound2, HCbound3, HCSplineplot} ) :
[> display( {VCbound1, VCbound2, VCbound3, VCSplineplot} ) :
[> display( {SLbound2, SLbound3, SLSplineplot} ) :
[> display( {TSbound1, TSbound2, TSbound3, TSSplineplot} ) :

```

[*Calculating planned period of motion along the circle trajectory*]

[*Horizontal circle:*]

```

[> HCTfShading := plot( 1 / InterpolyHC, θ1 = -π..π, 0..0.2, filled = true, color = orange,
    transparency = 0.8 ) :
[> HCTfBoarder := plot( 1 / InterpolyHC, θ1 = -π..π, 0..0.2, color = black, thickness = 2 ) :
[> display( [HCTfBoarder, HCTfShading] ) :
[> TfHC := evalf( int( 1 / InterpolyHC, θ1 = -3.14159..3.14159 ) )
    TfHC := 0.9178941914

```

(1)

[*Vertical circle:*]

```

[> VCTfShading := plot( 1 / InterpolyVC, θ2 = -π..π, 0..0.2, filled = true, color = orange,
    transparency = 0.8 ) :
[> VCTfBoarder := plot( 1 / InterpolyVC, θ2 = -π..π, 0..0.2, color = black, thickness = 2 ) :
[> display( [VCTfBoarder, VCTfShading] ) :
[> TfVC := evalf( int( 1 / InterpolyVC, θ2 = -3.14159..3.14159 ) )
    TfVC := 0.6407211965

```

(2)

[*Straight line:*]

```

> SLTfShading := plot(  $\frac{1}{\text{InterpolySL}}$ , x = 0.45 ..0.65, filled = true, color = orange, transparency
= 0.8 ) :
> SLTfBoarder := plot(  $\frac{1}{\text{InterpolySL}}$ , x = 0.45 ..0.65, color = black, thickness = 2 ) :
> display( [SLTfBoarder, SLTfShading] ) :
> TfSL := evalf( int(  $\frac{1}{\text{InterpolySL}}$ , x = 0.45 ..0.65 ) )
TfSL := 0.1428386517

```

(3)

Transition segment:

```

> TSTfShading := plot(  $\frac{1}{\text{InterpolyTS}}$ ,  $\theta t = -\pi ..\pi$ , 0 ..0.2, filled = true, color = orange, transparency
= 0.8 ) :
> TSTfBoarder := plot(  $\frac{1}{\text{InterpolyTS}}$ ,  $\theta t = -\pi ..\pi$ , 0 ..0.2, color = black, thickness = 2 ) :
> display( [TSTfBoarder, TSTfShading] ) :
> TfTS := evalf( int(  $\frac{1}{\text{InterpolyTS}}$ ,  $\theta t = -3.141592654 ..3.141592654$  ) ) :

```

Converting results to matlab code

Horizontal circle:

```

> #Matlab( $\phi$ 1, resultname = "phi1");
> #Matlab( $\phi$ 2, resultname = "phi2");
> #Matlab( $\phi$ 3, resultname = "phi3");
> #Matlab(D $\phi$ 1, resultname = "dphi1");
> #Matlab(D $\phi$ 2, resultname = "dphi2");
> #Matlab(D $\phi$ 3, resultname = "dphi3");
> #Matlab(DD $\phi$ 1, resultname = "ddphi1");
> #Matlab(DD $\phi$ 2, resultname = "ddphi2");
> #Matlab(DD $\phi$ 3, resultname = "ddphi3");

```

Vertical circle:

```

> #Matlab( $\sigma$ 1, resultname = "sigma1");
> #Matlab( $\sigma$ 2, resultname = "sigma2");
> #Matlab( $\sigma$ 3, resultname = "sigma3");
> #Matlab(D $\sigma$ 1, resultname = "dsigma1");
> #Matlab(D $\sigma$ 2, resultname = "dsigma2");
> #Matlab(D $\sigma$ 3, resultname = "dsigma3");
> #Matlab(DD $\sigma$ 1, resultname = "ddsigma1");
> #Matlab(DD $\sigma$ 2, resultname = "ddsigma2");

```

```
[> #Matlab(DD $\sigma$ 3, resultname = "ddsigma3");
```

[*Straight line:*

```
[> #Matlab( $\epsilon$ 1, resultname = "epsilon1");  
[> #Matlab( $\epsilon$ 2, resultname = "epsilon2");  
[> #Matlab( $\epsilon$ 3, resultname = "epsilon3");  
[> #Matlab(D $\epsilon$ 1, resultname = "depsilon1");  
[> #Matlab(D $\epsilon$ 2, resultname = "depsilon2");  
[> #Matlab(D $\epsilon$ 3, resultname = "depsilon3");  
[> #Matlab(DD $\epsilon$ 1, resultname = "ddepsilon1");  
[> #Matlab(DD $\epsilon$ 2, resultname = "ddepsilon2");  
[> #Matlab(DD $\epsilon$ 3, resultname = "ddepsilon3");
```

[*Transition segment:*

```
[> #Matlab( $\lambda$ 1, resultname = "lambda1");  
[> #Matlab( $\lambda$ 2, resultname = "lambda2");  
[> #Matlab( $\lambda$ 3, resultname = "lambda3");  
[> #Matlab(D $\lambda$ 1, resultname = "dlambda1");  
[> #Matlab(D $\lambda$ 2, resultname = "dlambda2");  
[> #Matlab(D $\lambda$ 3, resultname = "dlambda3");  
[> #Matlab(DD $\lambda$ 1, resultname = "ddlambd1");  
[> #Matlab(DD $\lambda$ 2, resultname = "ddlambd2");  
[> #Matlab(DD $\lambda$ 3, resultname = "ddlambd3");
```

Appendix

MATLAB: Simulink

Most of the plots presented in the paper are made in MATLAB, and the simulations are done by the use of Simulink. The most important Simulink systems are presented here. The complete set of files can be found in the file folder related to this thesis.

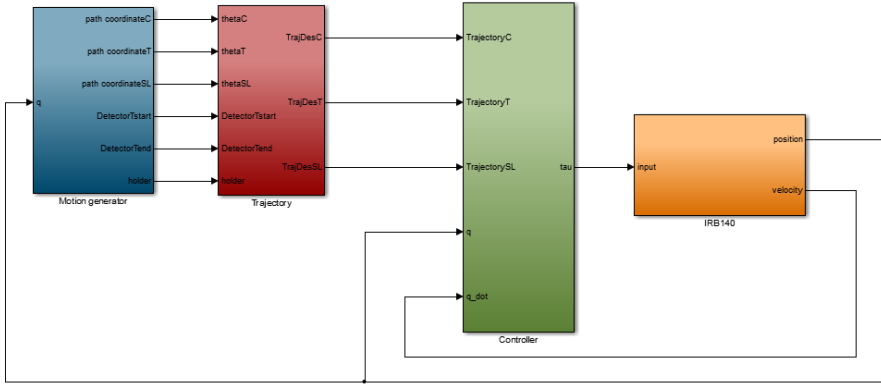


Figure C.1: Simulink model - Overall structure of the simulation of orbital stabilization on the motion planning cases

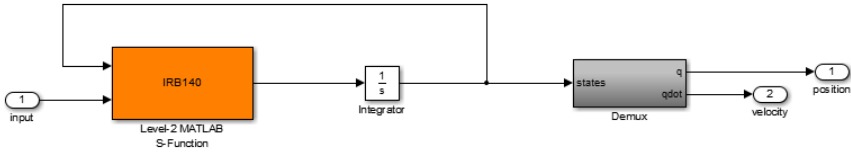


Figure C.2: IRB140 block - Containing a Level-2 MATLAB S-Function block for simulating the dynamics of the IRB140 manipulator.

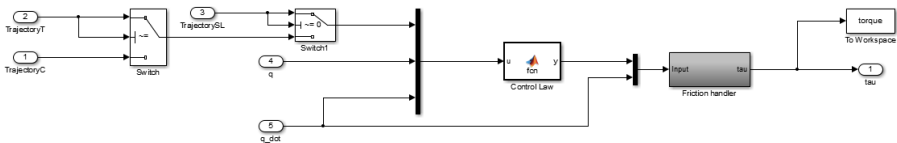


Figure C.3: Controller block - Taking in the desired and obtained joint values for orbital stabilization. Also containing a block for friction handling.

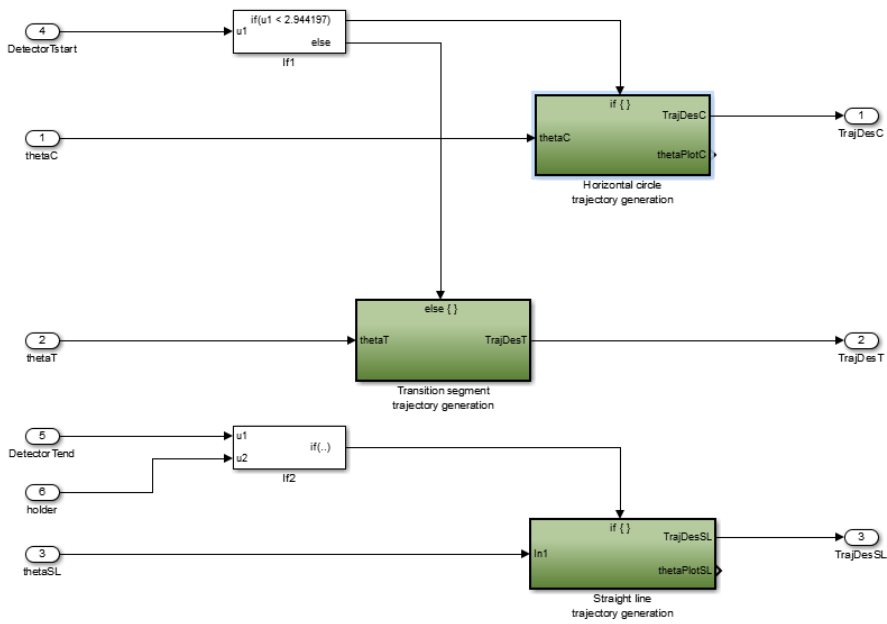


Figure C.4: Trajectory block - Structure of the trajectory generation, is generated in the Velocity profile block, the desired trajectory signal is then computed in a Matlab function.

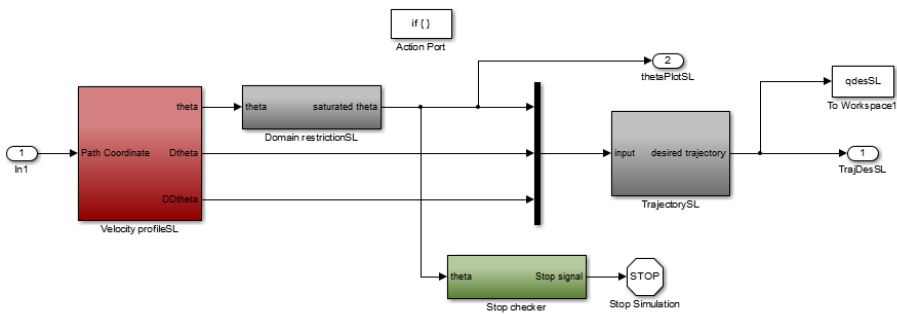


Figure C.5: Example of trajectory generation block for the straight line - Velocity profile is generated in the Velocity profile block, the desired trajectory signal is then computed in a Matlab function.

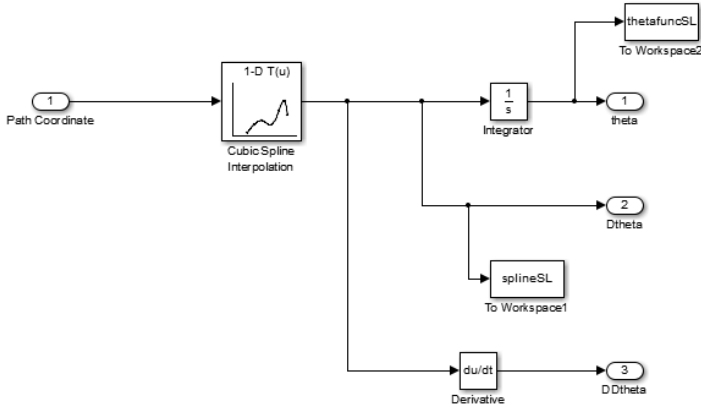


Figure C.6: Velocity profile block - Simple structure of the interpolation by the look-up-table block.

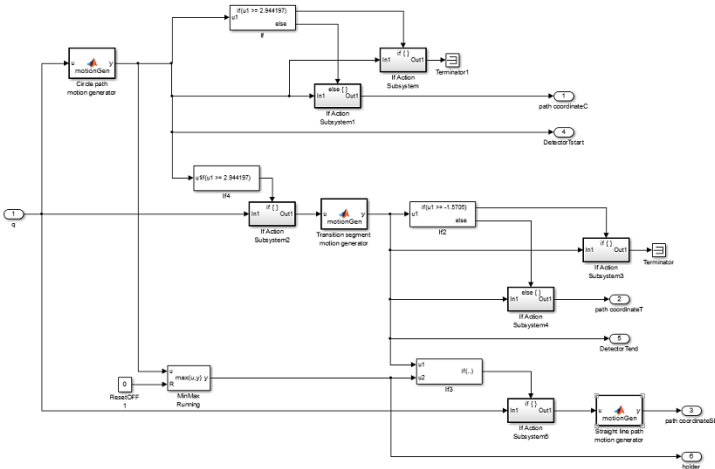


Figure C.7: Motion generator block structure - Taking the vector $q(t)$ as input.

Appendix **D**

MATLAB: Level-2 S-function

The manipulator was simulated using a Level-2 S-function. The information of this function is presented here. Some of the expressions in the function are very long, and only the shortened versions are seen here. The complete expressions can be seen in the files found in the file folder related to the thesis.

Table of Contents

Initialization	1
Register number of input and output ports	1
Setup functional port properties to dynamically inherited.	1
Hard-code certain port properties	1
Register a single dialog parameter	2
Set block sample time	2
Decide accelerator	2
Register a callback method for block	2
Set initial conditions	2
Input definitions	2
Mass values found from SolidWorks calculations	2
Inertia tensor matrices found by SolidWorks calculations	3
Vectors from frame i-1 to mass center of link i	3
Inertia Matrix Estimate	3
Gravity Vector Estimate	3
Coriolis & Centrifugal matrix Estimate	3
Defining the Dynamic System	4
Friction model	4
Calculating DDq from the dynamic model	4
Order reduction	4
Output definitions	4

Initialization

```
function IRB140(block)
    setup(block);

function setup(block)
```

Register number of input and output ports

```
block.NumInputPorts = 2;
block.NumOutputPorts = 1;
```

Setup functional port properties to dynamically inherited.

```
block.SetPreCompInpPortInfoToDynamic;
block.SetPreCompOutPortInfoToDynamic;
```

Hard-code certain port properties

```
block.InputPort(1).Dimensions = [6 1];
block.InputPort(2).Dimensions = [3 1];
block.InputPort(1).DirectFeedthrough = false;
block.InputPort(2).DirectFeedthrough = false;
```

```
block.OutputPort(1).Dimensions = [6 1];
```

Register a single dialog parameter

```
block.NumDialogPrms = 0;
```

Set block sample time

```
block.SampleTimes = [-1 0];
```

Decide accelerator

```
block.SetAccelRunOnTLC(false);
```

Register a callback method for block

```
block.RegBlockMethod('InitializeConditions', @InitializeConditions);  
block.RegBlockMethod('Outputs', @Outputs);
```

Set initial conditions

```
function InitializeConditions(block)  
block.InputPort(1).Data(1) = 0;  
block.InputPort(1).Data(2) = 0;  
block.InputPort(1).Data(3) = 0;  
block.InputPort(1).Data(4) = 0;  
block.InputPort(1).Data(5) = 0;  
block.InputPort(1).Data(6) = 0;  
  
function Outputs(block)
```

Input definitions

```
u1 = block.InputPort(2).Data(1);  
u2 = block.InputPort(2).Data(2);  
u3 = block.InputPort(2).Data(3);  
  
q1 = block.InputPort(1).Data(1);  
Dq1 = block.InputPort(1).Data(2);  
q2 = block.InputPort(1).Data(3);  
Dq2 = block.InputPort(1).Data(4);  
q3 = block.InputPort(1).Data(5);  
Dq3 = block.InputPort(1).Data(6);
```

Mass values found from SolidWorks calculations

```
m1 = 34.655;
```

```
m2 = 15.994;  
m3 = 20.862;
```

Inertia tensor matrices found by SolidWorks calculations

```
i11x = 0.51205253974; i11y = -0.385971e-2; i11z = 0.051305;  
i12x = 0.00136134; i12y = 0.46407468859; i12z = 0.0703356;  
i13x = 0.051305; i13y = -0.627205e-1; i13z = 0.55411384358;  
  
i21x = 0.0948179; i21y = -0.385971e-2; i21z = 0.037932;  
i22x = -0.385971e-2; i22y = 0.32860416324; i22z = -0.108897e-2;  
i23x = 0.037932; i23y = -0.108897e-2; i23z = 0.27746300488;  
  
i31x = 0.50006091595; i31y = -0.186325e-2; i31z = 0.000934876;  
i32x = -0.186325e-2; i32y = 0.0751527; i32z = -0.152041e-1;  
i33x = 0.000934876; i33y = -0.152041e-1; i33z = 0.51542475434;
```

Vectors from frame i-1 to mass center of link i

```
r0c1x = 0.08903; r0c1y = -0.2789e-1; r0c1z = 0.04312;  
r1c2x = 0.19829; r1c2y = -0.9243e-1; r1c2z = 0.00973;  
r2c3x = 0.07996; r2c3y = 0.00456; r2c3z = 0.00586;
```

Inertia Matrix Estimate

```
m11 = 0.5000000000e0 * i32y + 0.5000000000e0 * i31x + 0.5000000000e0 * i22...  
m12 = -0.1000000000e1 * i31z * sin(q3 + q2) - 0.1000000000e1 * i32z * cos(...  
m13 = 0.1000000000e1 * r2c3x * m3 * r2c3z * sin(q3 + q2) + 0.1000000000e1 ...  
m21 = 0.1000000000e1 * r1c2y * m2 * r1c2z * sin(q2) + 0.1000000000e1 * r2c...  
m22 = r2c3y * m3 * cos(q3) * (-0.719999999999999973e0) + r2c3x * m3 * sin(...  
m23 = 0.1000000000e1 * i33z + 0.1000000000e1 * m3 * r2c3x ^ 2 + r2c3y * m3...  
m31 = 0.1000000000e1 * r2c3y * m3 * r2c3z * cos(q3 + q2) + 0.1000000000e1 ...  
m32 = 0.1000000000e1 * i33z + 0.1000000000e1 * m3 * r2c3y ^ 2 - 0.36000000...  
m33 = 0.1000000000e1 * i33z + 0.1000000000e1 * m3 * r2c3y ^ 2 + 0.10000000...
```

Gravity Vector Estimate

```
g = 9.81;  
g1 = 0;  
g2 = 0.1000000000e1 * r2c3y * m3 * sin(q3 + q2) * g - 0.1000000000e1 * r1c...  
g3 = -0.1000000000e1 * r2c3x * m3 * cos(q3 + q2) * g + 0.1000000000e1 * r2...
```

Coriolis & Centrifugal matrix Estimate

```
c11 = sin(q2) * m3 * Dq1 * r2c3z * (-0.277555756156289135e-16) + ((0.10000...  
c12 = (-0.1000000000e1 * cos(q2) * i22z - 0.1000000000e1 * cos(q3 + q2) * ...  
c13 = (-0.1000000000e1 * cos(q3 + q2) * i31z - 0.1000000000e1 * r2c3z * m3...  
c21 = (-0.5000000000e0 * m2 * r1c2x ^ 2 * sin((2 * q2)) - 0.7000000000e-1 ...
```

```

c22 = (r2c3y * m3 * sin(q3) * 0.71999999999999973e0 + r2c3x * m3 * cos(q3...
c23 = (r2c3x * m3 * cos(q3) * (-0.35999999999999987e0) + r2c3y * m3 * sin...
c31 = (0.180000000000e0 * r2c3x * m3 * cos(q3) - 0.180000000000e0 * r2c3x * m3...
c32 = (-0.360000000000e0 * r2c3y * m3 * sin(q3) + 0.360000000000e0 * r2c3x * m...

c33 = 0;

```

Defining the Dynamic System

```

M = [m11 m12 m13;m21 m22 m23;m31 m32 m33];
C = [c11 c12 c13; c21 c22 c23; c31 c32 c33];
G = [g1; g2; g3];

```

Friction model

```

Ks1 = 21.522476573748172;
Ks2 = 26.685483301181058;
Ks3 = 9.699584289009023;
Kp1 = 9.193839979114159;
Kp2 = 5.106491748638457;
Kp3 = 3.765788262834416;

Fv1 = Ks1*sign(Dq1) + Kp1 * Dq1;
Fv2 = Ks2*sign(Dq2) + Kp2 * Dq2;
Fv3 = Ks3*sign(Dq3) + Kp3 * Dq3;

```

Calculating DDq from the dynamic model

```

DynSys = (M^(-1))*([u1-Fv1; u2-Fv2; u3-Fv3] - C*[Dq1;Dq2;Dq3] - G);

```

Order reduction

```

dx1 = Dq1;
dx2 = DynSys(1);
dx3 = Dq2;
dx4 = DynSys(2);
dx5 = Dq3;
dx6 = DynSys(3);

```

Output definitions

```

dx = [dx1;dx2;dx3;dx4;dx5;dx6];
block.OutputPort(1).Data = dx;

```

Published with MATLAB® R2013a

Appendix **E**

ABB Datasheet

Specifications for the IRB140, as presented by ABB, are included here in a compressed form. A more complete datasheet is included in the file folder related to the thesis.

IRB 140 Industrial Robot

Main Applications

Arc welding
Assembly
Cleaning/Spraying
Machine tending
Material handling
Packing
Deburring



Small, Powerful and Fast

Compact, powerful IRB 140 industrial robot.

Six axis multipurpose robot that handles payload of 6 kg, with long reach (810 mm). The IRB 140 can be floor mounted, inverted or on the wall in any angle. Available as Standard, FoundryPlus, Clean Room and Wash versions, all mechanical arms completely IP67 protected, making IRB 140 easy to integrate in and suitable for a variety of applications. Uniquely extended radius of working area due to bend-back mechanism of upper arm, axis 1 rotation of 360 degrees even as wall mounted.

The compact, robust design with integrated cabling adds to overall flexibility. The Collision Detection option with full path retraction makes robot reliable and safe.

Using IRB 140T, cycle-times are considerably reduced where axis 1 and 2 predominantly are used.

Reductions between 15-20 % are possible using pure axis 1 and 2 movements. This faster versions is well suited for packing applications and guided operations together with PickMaster.

IRB Foundry Plus and Wash versions are suitable for operating in extreme foundry environments and other harsh environments with high requirements on corrosion resistance and tightness. In addition to the IP67 protection, excellent surface treatment makes the robot high pressure steam washable. The white-finish Clean Room version meets Clean Room class 10 regulations, making it especially suited for environments with stringent cleanliness standards.

IRB 140

Specification

Robot versions	Handling capacity	Reach of 5th axis	Remarks
IRB 140/IRB 140T	6 kg	810 mm	
IRB 140F/IRB 140TF	6 kg	810 mm	Foundry Plus Protection
IRB 140CR/ IRB 140TCR	6 kg	810 mm	Clean Room
IRB 140W/ IRB 140TW	6 kg	810 mm	Wash Protection
Supplementary load (on upper arm alt. wrist)			
on upper arm	1 kg		
on wrist	0.5 kg		
Number of axes			
Robot manipulator		6	
External devices		6	
Integrated signal supply		12 signals on upper arm	
Integrated air supply		Max. 8 bar on upper arm	

Performance

Position repeatability 0.03 mm (average result from ISO test)

Axis movement	Axis	Working range
	1, C Rotation	360°
	2, B Arm	200°
	3, A Arm	280°
	4, D Wrist	Unlimited (400° default)
	5, E Bend	240°
	6, P Turn	Unlimited (800° default)

Max. TCP velocity 2.5 m/s

Max. TCP acceleration 20 m/s²

Acceleration time 0-1 m/s 0.15 sec

Velocity

Axis no.	IRB 140	IRB 140T
1	200°/s	250°/s
2	200°/s	250°/s
3	260°/s	260°/s
4	360°/s	360°/s
5	360°/s	360°/s
6	450°/s	450°/s

Cycle time

5 kg Picking side	IRB 140	IRB 140T
cycle 25 x 300 x 25 mm	0,85s	0,77s


Electrical Connections

Supply voltage	200–600 V, 50/60 Hz
Rated power	
Transformer rating	4.5 kVA
Power consumption typicly	0,4 kW

Physical

Robot mounting	Any angle
Dimensions	
Robot base	400 x 450 mm
Robot controller H x W x D	950 x 800 x 620 mm
Weight	
Robot manipulator	98 kg

Environment

Ambient temperature for	
Robot manipulator	5 – 45°C
Relative humidity	Max. 95%
Degree of protection,	
Manipulator	IP67
Options	
	Foundry
	Wash (High pressure steam washable)
	Clean Room, class 6 (certified by IPA)
	
Noise level	Max. 70 dB (A)
Safety	
	Double circuits with supervision, emergency stops and safety functions, 3-position enable device
Emission	EMC/EMI-shielded

Data and dimensions may be changed without notice

Working range

