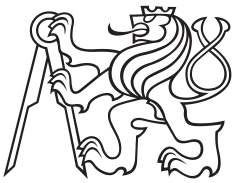
Bachelor thesis



Czech Technical University in Prague

F2

Faculty of Mechanical Engineering
Department of Mechanics, Biomechanics and Mechatronics

Trajectory planning for 5-axis hybrid parallel-serial Delta robot

Filip Zítek

Supervisor: Ing. Petr Beneš, Ph.D.
Field of study: Theoretical Fundamentals of Mechanical Engineering
August 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Zítek** Jméno: **Filip** Osobní číslo: **483604**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav mechaniky, biomechaniky a mechatroniky**

Studijní program: **Teoretický základ strojního inženýrství**

Studijní obor: **bez oboru**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Plánování trajektorie pro 5-osý hybridní paralelní-sériový delta robot

Název bakalářské práce anglicky:

Trajectory planning for 5-axis hybrid parallel-serial delta robot

Pokyny pro vypracování:

1. Seznamte se s kinematikou paralelních robotů typu delta včetně jejich hybridních modifikací.
2. Nastudujte řešení přímé a inverzní kinematické úlohy.
3. Připravte plánování trajektorie s využitím PH (Pythagorean-Hodograph) křivek.
4. Funkčnost vytvořeného plánovače trajektorie ověřte experimentálně.

Seznam doporučené literatury:

[1] Su, T., Cheng, L., Wang, Y.,Liang, X.,Zheng, J., Zhang, H.: Time-Optimal Trajectory Planning for Delta Robot Based on Quintic Pythagorean-Hodograph Curves, IEEE Access, vol. 6, pp. 28530-28539, 2018
[2] Merlet, J.P.: Parallel Robots. Solid Mechanics and Its Applications, vol. 128, Heidelberg, Springer 2006
[3] Kuřina, J.: Plánování trajektorie paralelního robota, bakalářská práce, FS ČVUT v Praze, 2017

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Petr Beneš, Ph.D. odbor mechaniky a mechatroniky FS

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **22.04.2022** Termín odevzdání bakalářské práce: **15.08.2022**

Platnost zadání bakalářské práce: _____

Ing. Petr Beneš, Ph.D. podpis vedoucí(ho) práce	prof. Ing. Michael Valášek, DrSc. podpis vedoucí(ho) ústavu/katedry	doc. Ing. Miroslav Španiel, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Annotation

Author:	Filip Zítek
Thesis title in English:	Trajectory planning for 5-axis hybrid parallel-serial Delta robot
Thesis title in Czech:	Plánování trajektorie pro 5-osý hybridní paralelní-sériový Delta robot
Academic year:	2021/2022
Department/Division:	Department of Mechanics, Biomechanics and Mechatronics, Division of Mechanics and Mechatronics
Supervisor:	Ing. Petr Beneš, Ph.D.
Bibliografic information:	Number of pages: 55 Number of figures: 24 Number of attachments: 1× CD
Keywords:	trajectory planning, Delta robot, hybrid kinematic structure, Pythagorean-Hodograph curves
Klíčová slova:	plánování trajektorie, Delta robot, hybridní kinematická struktura, Pythagorean-Hodograph křivky
Abstract:	This bachelor thesis deals with trajectory planning for a 5-axis hybrid parallel-serial Delta robot. The task is to prepare a planner using Pythagorean-Hodograph curves for blends between lines. First, the kinematics of the robot is studied. After that, the Pythagorean-Hodograph curve for blends is derived, and the planner is created. In addition, a user interface is provided for easier use of the planner. Finally, the functionality of the trajectory planner is verified experimentally.
Abstrakt:	Tato bakalářská práce se zabývá plánováním trajektorie pro 5-osý hybridní paralelní-sériový Delta robot. Úkolem je připravit plánovač s využitím Pythagorean-Hodograph křivek pro napojení úseček. Nejprve je nastudována kinematika tohoto robotu. Poté je odvozena Pythagorean-Hodograph křivka pro napojování a je vytvořen plánovač, spolu s uživatelským rozhraním pro snazší ovládání. Na závěr je funkčnost vytvořeného plánovače ověřena experimentálně.

Acknowledgements

I would like to thank my supervisor Ing. Petr Beneš, Ph.D., for guiding my bachelor thesis. I am also grateful to Testbed for Industry 4.0 for the opportunity to work on this topic. Specifically, I would like to thank Baran Alikoç, Ph.D. and Ing. Vojtěch Šustr for providing consultations and valuable suggestions and Bc. Václav Kubáček for helping me with the experiment. Thanks should also go to my family and friends, especially Bc. Šárka Trhoňová, for their tremendous support.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, August 9, 2022

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 9. srpna 2022

Abstract

This bachelor thesis deals with trajectory planning for a 5-axis hybrid parallel-serial Delta robot. The task is to prepare a planner using Pythagorean-Hodograph curves for blends between lines. First, the kinematics of the robot is studied. After that, the Pythagorean-Hodograph curve for blends is derived, and the planner is created. In addition, a user interface is provided for easier use of the planner. Finally, the functionality of the trajectory planner is verified experimentally.

Keywords: trajectory planning, Delta robot, hybrid kinematic structure, Pythagorean-Hodograph curves

Supervisor: Ing. Petr Beneš, Ph.D.
Odbor mechaniky a mechatroniky,
Technická 4,
160 00 Praha 6

Abstrakt

Tato bakalářská práce se zabývá plánováním trajektorie pro 5-osý hybridní paralelní-sériový Delta robot. Úkolem je připravit plánovač s využitím Pythagorean-Hodograph křivek pro napojení úseček. Nejprve je nastudována kinematika tohoto robotu. Poté je odvozena Pythagorean-Hodograph křivka pro napojování a je vytvořen plánovač, spolu s uživatelským rozhraním pro snazší ovládání. Na závěr je funkčnost vytvořeného plánovače ověřena experimentálně.

Klíčová slova: plánování trajektorie, Delta robot, hybridní kinematická struktura, Pythagorean-Hodograph křivky

Překlad názvu: Plánování trajektorie pro 5-osý hybridní paralelní-sériový Delta robot

Contents

Figures

Tables

Chapter 1

Introduction

Automation of manufacturing processes is a common trend in the industry. For that, various types of robots are used depending on the application. For pick-and-place operations, a Delta robot is often used. In its standard form, it has only three actuated axes, and the end-effector cannot be rotated. A hybrid modification of this robot can be created by adding a serial wrist to the parallel structure. Two additional degrees of freedom are gained this way, and the end-effector can now change its orientation in space.

One of these hybrid Delta robots is located in the Testbed for Industry 4.0 at the Czech Institute of Informatics, Robotics and Cybernetics (CIIRC), where it serves research purposes. However, the current solution for trajectory planning and control of this robot cannot handle all five axes simultaneously. Because of that, the wrist part of the robot has been fixed in its position up to now, restricting the application possibilities. Furthermore, only three pre-programmed modes of operation are available, and there is no easy way to generate a custom trajectory based on the needs of the user.

To use the full potential of the robot's structure and kinematics, for example, to use all five axes at the same time and be able to plan the necessary movement, a trajectory planner must be created.

The aim of this thesis is to design such a trajectory planner. For that, a kinematic model of the robot has to exist, as well as a solution for forward and inverse kinematics. Thus, the suggested guidelines for this thesis are as follows.

- Familiarize yourself with the kinematics of parallel robots of type Delta, including their hybrid modifications
- Study the forward and inverse kinematics
- Prepare trajectory planning using PH (Pythagorean-Hodograph) curves
- Verify the functionality of the created trajectory planner experimentally

Both the kinematic model and the forward and inverse kinematics are already existing for our Delta robot. Therefore, the task is to study them and learn how to use them. The trajectory planner should, in addition to the requirements mentioned above, also be user-friendly so that anyone can work with the robot. The definition of points in the workspace should be done in a way that would

be reasonable for a human to work with. However, at the same time, it should be easily implementable in some higher-level motion planning system, which would decide, for instance, based on computer vision, where the robot should move. The movement along the generated trajectory should be as fast as possible while respecting defined kinematic limits and remaining smooth to minimize the vibrations of the mechanical structure.

Blending curves are used to shorten the path and eliminate the need to stop when changing the direction of movement. Thus, the total duration of the trajectory is shorter. In the current solution, arcs are used for the blending, but that creates a discontinuous curvature along the path and causes vibrations. In recent years, Pythagorean-Hodograph curves have been examined for the purpose of blending curves. In our case, existing solutions were insufficient, so a modified version was introduced.

To make the trajectory planner easy to use, a user interface with a visual representation of the trajectory was created. As a consequence, it is not necessary for the user to have programming skills.

This thesis is organized as follows. Chapter 2 gives a brief overview of the different types of robots, shows some applications of Delta robots, and then describes in detail the hybrid Delta robot for which the planner is created. In Chapter 3, the blending curve is derived, and the trajectory planner with the user interface is designed. The experimental verification of the planner is made in Chapter 4. Chapter 5 concludes the thesis.

The trajectory planner was created in Matlab R2021a. All of the components of the planner are available as a code in the attachments.

Chapter 2

Serial, parallel and hybrid robots

The most common type of kinematic structure of robots used in industry is serial. Individual parts of such a robot are connected to adjacent bodies by joints with exactly one actuated degree of freedom (DoF), for instance, revolute or prismatic [1]. The distinctive feature of serial robots is an open kinematic chain. On the contrary, parallel robots are mechanisms that contain a closed-loop kinematic chain [1].

First, a few terms have to be explained. An operational space, also called a task space, is the space where the robot performs its movement [2]. A workspace is a subset of operational space that the robot can reach [2]. A joint space, also called a configuration space, represents all possible values of actuator positions [2].

Repeatability is the maximum distance between the points reached for the same desired position of the end-effector [1]. Absolute accuracy is the distance between the desired final position and the actual final position [1].

2.1 Comparison of different kinematic structures

In general, serial robots are less precise than parallel robots [3]. Although the repeatability of serial robots is decent, the absolute accuracy is poor [1]. The reason is that all errors, such as the flexure of the individual links, geometrical tolerances of joints, or backslashes in gearboxes, accumulate along the kinematic structure. Even a tiny error in every joint or link causes a significant error in the position of the robot's end-effector [1]. Parallel robots are therefore more precise since the errors are not added cumulatively.

Another disadvantage of serial robots is that every link has to hold, in addition to the load mass, the weight of the following bodies up to the end-effector [3]. Thus, individual links are experiencing a bending moment. Because of that, they need to be stiffened and are becoming even heavier [1]. Parallel robots are more rigid because at least some links are subject to only tension or compression forces, not a bending moment, and the load is distributed in multiple chains. Furthermore, the links do not have to support the weight of the whole structure. Heavy parts, such as motors, are fixed to the stationary frame. Therefore, the total possible load capacity is considerably greater, and the mechanism can be more subtle [4].

This leads to another difference between serial and parallel structures. The heavy bodies of serial robots experience high inertial forces, and thus velocity is

limited, whereas parallel robots can operate with much higher velocities due to lightweight and low-inertia bodies [3].

On the other hand, the workspace of a parallel robot is much smaller than that of a serial robot of similar size [4]. This is caused by the mechanical structure of the parallel robot. Additional factors, such as the limits of actuated and passive joints or possible self-collisions of individual bodies, can further reduce the workspace [1].

Another disadvantage of parallel structures is their complex kinematics [4]. Especially the inverse kinematics task is rather difficult, as can be seen in Section 2.3.1. In addition, prospective singular configurations of parallel structures create additional degrees of freedom that are not controllable and can lead to fatal situations [4].

Hybrid robots combine serial and parallel structures. It extends the possible applications while minimizing the disadvantages of purely serial or parallel structures. An example of a hybrid robot is a 5-axis Delta robot, described in detail in Section 2.3. A serial wrist arm is mounted on the movable platform of a standard parallel Delta robot. Hence, two additional degrees of freedom are added, allowing the end-effector also to rotate. With that, more sophisticated manipulation tasks can be performed. Furthermore, the workspace of this hybrid Delta robot is larger due to the added wrist part.

2.2 Applications of Delta robots

Applications for different robots are given by their properties, advantages, and disadvantages, which are described above. For a Delta robot, the key feature is fast and very precise movement. Therefore, the main application is in pick-and-place operations, where the task is to pick up an object, usually from some conveyor belt, and place that object somewhere else, for instance, on another conveyor belt, in the case of a sorting application, directly into some container or box, in the case of a packaging application, or placing that object in some assembly unit, in the case of an assembly application [3].

The robotic cell, or directly the end-effector of the robot, is often equipped with a camera that detects individual objects on the conveyor belt and sends their positions to the trajectory planner [3]. The hybrid Delta robot, with the serial wrist and the ability to rotate the object in space during pick-and-place operations, can even place the object with the desired orientation. This opens up packaging and assembly applications for all kinds of industries. The most common are the medical, pharmaceutical, food processing, and microchip assembly industries [5].

The fast movement of Delta robots allows high speeds of the conveyor belts under it while still being able to pick and place objects precisely. These robots can perform more than 100 pick-and-place operations per minute, which is much faster than a human [5]. Thus, automation of packaging and assembly processes is economically advantageous, and Delta robots are being installed more and more frequently [3].

2.3 5-axis hybrid Delta robot

The trajectory planner created in this thesis is designed specifically for a Delta robot located in the Testbed for Industry 4.0 at the Czech Institute of Informatics, Robotics and Cybernetics (CIIRC), Czech Technical University (CTU) in Prague[1]. It is a hybrid Delta robot with five axes and a nominal payload of 6 kilograms [6]. The robot is shown in Fig. 2.1.

Figure 2.1: 5-axis Delta robot in Testbed for Industry 4.0 at CIIRC CTU

The robot consists of an upper part with parallel arms and a lower part with a serial wrist, as can be seen in Fig. 2.2. The three actuated revolute joints of the upper parallel arm axes, denoted A1, A2, and A3, are attached to the basic frame on top of the robot. Each parallel arm consists of an upper arm and a pair of forearm links that form a parallelogram. All three arms are connected to a movable platform.

The two additional actuators for axes A4 and A5 are also fixed to the basic frame. They are connected via two coaxial telescopic Cardan shafts to the movable platform. The outer shaft is for the A4 axis and is connected to the worm gearbox below the movable platform (see Fig. 2.2). The inner shaft of the A5 axis is connected to the worm drive, which transforms the relative rotation between the inner and outer shafts into the tilt of the swivel part attached to the worm gearbox.

The three upper arms cause a purely translational motion of the movable platform, as in the case of a standard 3-DoF Delta robot. The A4 axis allows

[1] `https://ricaip.eu/testbed-prague/`

rotation of the worm gearbox around a vertical axis, as shown in a kinematic model in Fig. 2.3. The A5 axis then allows rotation of the swivel part with an attached tool around a horizontal axis, which is also shown in Fig. 2.3.

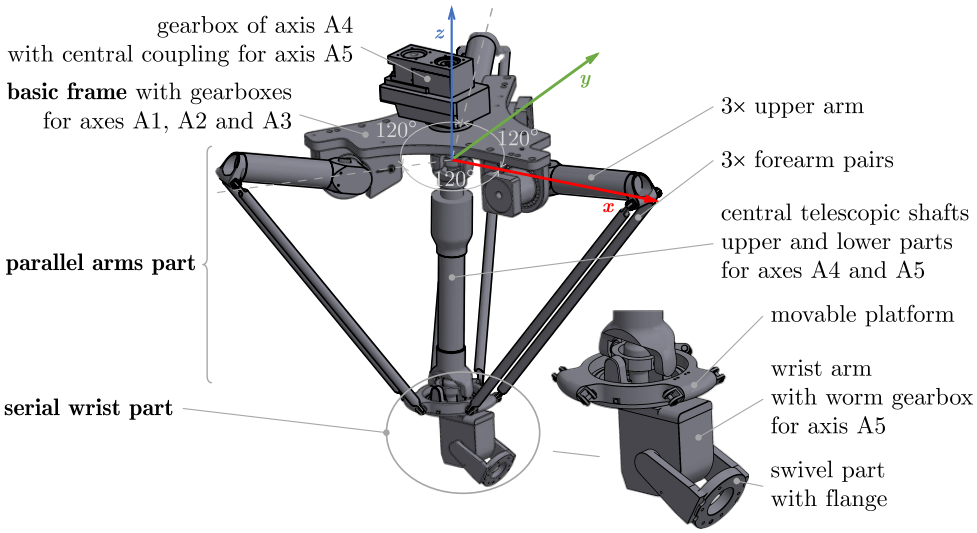


Figure 2.2: 3D model of the 5-axis Delta robot

A kinematic model of this robot (see Fig. 2.3) was made by simplifying the individual bodies of the robot into lines. The global coordinate frame of this mechanism is also shown in the figure. Its origin is located in the center of the upper universal joint of the telescopic Cardan shafts. The x axis of this coordinate frame points towards the A1 actuator and is horizontal, z axis points upwards.

Important joints of the links were named as follows. Actuated joints of the three parallel arms are called $\mathbf{A}_i$, $i = 1, 2, 3$. Elbow joints, where the upper arm meets the forearm, are called $\mathbf{B}_i$. The point where the forearm connects to the movable platform is $\mathbf{C}_i$, and the center of this platform is point $\mathbf{D}$. Point $\mathbf{F}$ is where the center of the worm wheel is and, therefore, where the swivel part is attached to the worm drive. Point $\mathbf{E}$ has the same z coordinate as point $\mathbf{D}$ but is shifted to be above point $\mathbf{F}$.

At the end of the swivel part, in the center of the flange, is the point $\mathbf{H}$. If a tool is attached to this flange, a Tool Center Point (TCP) is defined at the end of the tool. That is the point that needs to be defined in the workspace to perform a pick or place operation. Thus, the trajectory planner in Chapter 3 will generate a trajectory for the TCP.

2.3.1 Forward and inverse kinematics

The position and orientation of the robot's end-effector (gripper or some other tool) are related to the positions of the actuated joints through the kinematic structure of the robot. Since we are dealing with a 5-DoF robot, the easiest way to define the orientation of the end-effector is directly by the position of the axes in the wrist of the robot. Therefore, the pose (combination of position and orientation)

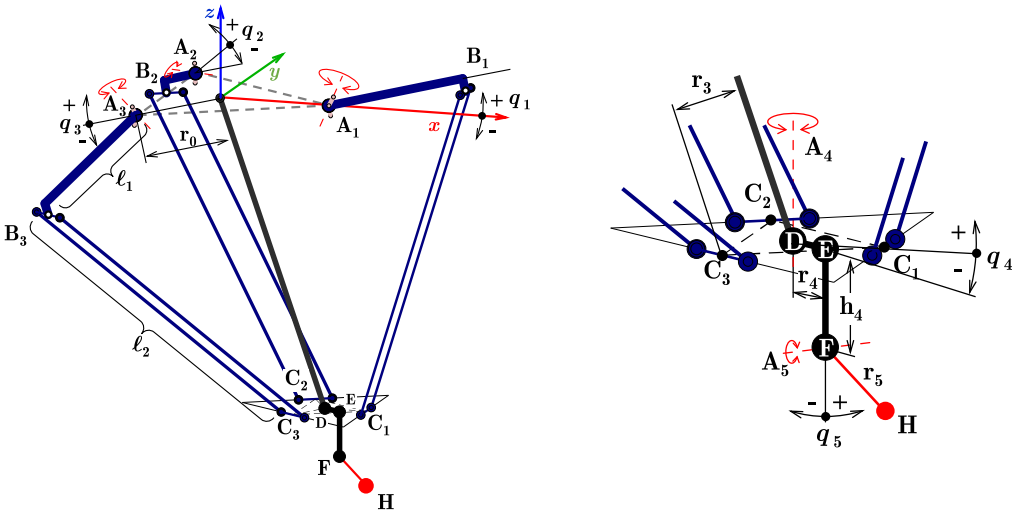


Figure 2.3: Kinematic model of the 5-axis Delta robot (left) and a detail of the movable platform and serial wrist part (right)

of the end-effector is defined by $[x_H, y_H, z_H, q_4, q_5]$. The related positions of the actuators in the joint space are $[q_1, q_2, q_3, q_4, q_5]$. The task of transforming the joint-space coordinates (positions and their time derivatives) into the task-space coordinates is called forward kinematics. The opposite, the task of transforming from the task space to the joint space, is called inverse kinematics [2].

For a standard 3-DoF Delta robot, this is a problem that was already solved, for example, in [3]. For a serial robot, solutions can be found in almost all robotics textbooks, for example, in [2] or [7]. For our hybrid robot, forward and inverse kinematics can be solved by separating the upper parallel arms part and the serial wrist part. Then the already existing solutions can be easily applied to our robot. This was already done in [8], so the derivation of forward and inverse kinematics is not part of my thesis. A brief overview of the derivation in [8] is given in the following paragraphs.

Forward kinematics

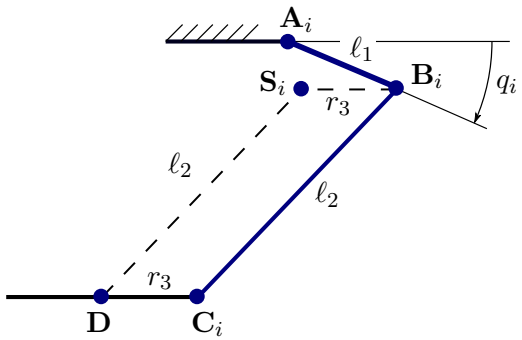


Figure 2.4: Side view on one of the parallel arms of Delta robot

First, the parallel arms part will be discussed. The task is to get the point **D**

with known points $\mathbf{A}_i$ and the position of the axes q_i, $i = 1, 2, 3$ for the individual axes. Point $\mathbf{B}_i$ can be calculated from point $\mathbf{A}_i$, angle q_i, and length of the upper arm ℓ_1. Then, a point $\mathbf{S}_i = [x_{S_i}, y_{S_i}, z_{S_i}]^{\mathrm{T}}$ is created by translating point $\mathbf{B}_i$ by the vector $\overrightarrow{\mathbf{C}_i\mathbf{D}}$. The orientation of this vector is known and the length is r_3 (see Fig. 2.4). After that, for the three upper arms, a sphere is made with a center at the point $\mathbf{S}_i$ and a radius ℓ_2. The point $\mathbf{D} = [x_D, y_D, z_D]^{\mathrm{T}}$ can be found as an intersection of the three spheres by solving the following set of equations:

$$(x_D - x_{S_i})^2 + (y_D - y_{S_i})^2 + (z_D - z_{S_i})^2 = \ell_2^2 \tag{2.1}$$

There are two solutions to this equation set, with two different values for z_D. The lower value is chosen as point $\mathbf{D}$ is below the base frame of the robot.

In the serial wrist part, the task is to get from point $\mathbf{D}$ to point $\mathbf{H}$ with known joint positions q_4, q_5 and lengths r_4, h_4, r_5. This can be done using the following equation:

$$\mathbf{H} = \mathbf{D} + \begin{bmatrix} (r_4 + r_5 \sin q_5) \cos q_4 \\ (r_4 + r_5 \sin q_5) \sin q_4 \\ -h_4 - r_5 \cos q_5 \end{bmatrix} \tag{2.2}$$

If a tool is attached to the flange of the robot, the length of the tool can be added to the length r_5. Then, a TCP is computed instead of the $\mathbf{H}$ point of the robot. This can be done only for tools that do not offset the TCP from the line between points $\mathbf{F}$ and $\mathbf{H}$, as in the case of our gripper. In other cases, a more complex transformation would be needed.

Inverse kinematics

The task is to obtain the joint positions q_1, q_2, q_3 from the known position of the point $\mathbf{H}$ and the joints q_4, q_5. The serial wrist part can be solved using equation (2.2) to obtain the point $\mathbf{D}$. If a tool is attached, the same principle as described in the forward kinematics section can be applied.

Then, for each of the three parallel arms, a vector $\overrightarrow{\mathbf{C}_i\mathbf{B}_i}$, $i = 1, 2, 3$, is used. The coordinates of point $\mathbf{C}_i$ are expressed using the coordinates of point $\mathbf{D}$ and distance r_3, and the coordinates of point $\mathbf{B}_i$ using the lengths r_0, ℓ_1 and angle q_i. Since the length of the vector $\overrightarrow{\mathbf{C}_i\mathbf{B}_i}$ must be ℓ_2, one can get kinematic constraints:

$$f_i(q_i, x_D, y_D, z_D) = 0 \tag{2.3}$$

After simplification and rearranging, the constraints with unknown q_i can be expressed in the form

$$a_i \cos q_i + b_i \sin q_i + c_i = 0. \tag{2.4}$$

After applying trigonometric identities, the joint positions can be found by solving the quadratic equations

$$a_i \frac{1 - t_i^2}{1 + t_i^2} + b_i \frac{2t_i}{1 + t_i^2} + c_i = 0, \tag{2.5}$$

where the joint positions are $q_i = 2 \arctan(t_i)$. The quadratic equation (2.5) has two roots. The one that creates the robot configuration with the joint $\mathbf{B}_i$ pointing out of the mechanism is chosen [3].

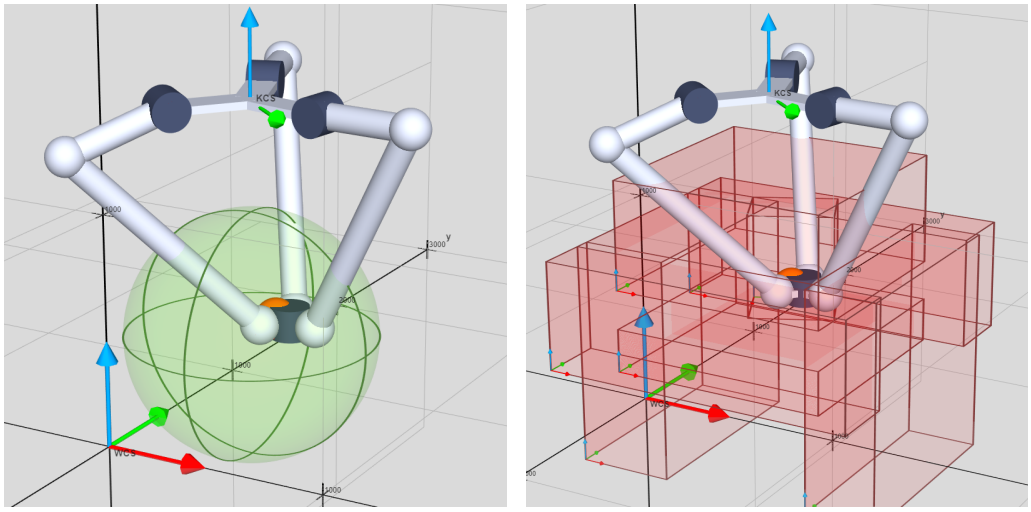
2.3.2 Workspace

The workspace is defined for the robot's point **E** (see Fig. 2.3) because the Programmable Logic Controller (PLC) that controls the robot cannot handle the kinematics of the five axes and assumes that A5 is fixed in the original position. Therefore, the point that is checked in the workspace must be somewhere on the robot, where the robot's model in the PLC (shown in Fig. 2.5) corresponds to the real robot.

To be in the workspace, the point **E** must be in the green sphere in Fig. 2.5a, which represents the work zone. Another condition is that the point cannot be in any of the blocked zones, represented by red blocks in Fig. 2.5b.

These blocked zones are added for the following reasons. One block is placed on each side of the robot to avoid collisions with the walls of the cage around the robot. Two blocks restrict movement in the vertical direction to not collide with the conveyor system under the robot and not exceed the stroke limit of the telescopic shafts. One more blocked zone is added to avoid collision with the tool holder placed in the work zone since it is not used yet.

The trajectory planner, created in Section 3, checks whether the point **E** is in the defined workspace at every point along the path or not. In the latter case, the user is notified that the desired trajectory is outside of the workspace and must modify it. Due to that, the trajectory that is generated always passes through the check in the PLC.



(a) : Work zone **(b) :** Blocked zones

Figure 2.5: Workspace of the Delta robot

Chapter 3

Trajectory planning

The aim of trajectory planning is to generate inputs to the robot's actuators so that the robot's end-effector will follow the desired trajectory [4]. A trajectory is a combination of path and motion law [2].

A path is a purely geometrical description of the motion of the end-effector. The path may be too complex to be fully specified by the user; thus, the user specifies only the extremal points, intermediate points, or geometric primitives, and the path is generated according to that [7]. Individual parts of the path, defined by some points, are called segments. From a mathematical point of view, a path is a parametric curve (or multiple curves for individual segments) $\mathbf{X}(p)$ for some parameter p. It is convenient to use the arc length parameter s, as it describes the position along the path [7]. So, the parametrization $\mathbf{X}(s)$ will often be used.

In some cases, position and orientation are defined separately because then the path is easier to define, and orientation is important only at the endpoints, as it is in pick-and-place applications [9]. The case for coupled position and orientation is, for example, in welding or milling robots, where many via points have to be interpolated, and the tool's orientation is essential at every point on the path [9].

The orientation can be generally defined by three angles (for example, Euler angles, or roll, pitch, yaw), but since our robot has only five degrees of freedom, the easiest way for the user to define the orientation of the tool in space is by the joint positions of the wrist part q_4 and q_5 (shown in Fig. 2.3). Paths can be generated either in joint space or in operational space. When interpolating between some points in joint space, the resulting path in operational space is not predictable, and the robot may collide with some obstacle [2]. Due to the very restricted workspace of Delta robots, the chance of collision is rather high and, therefore, paths are usually generated in the operational space.

A time scaling function $s(t)$, also called motion law, specifies the time instances when the individual points on the path are reached [2]. This function is determined by the user's requirements and by satisfying various constraints.

For example, the robot's actuators have physical limits that cannot be exceeded. Those limits are of two types – kinematic and dynamic [9]. Kinematic limits involve a maximum value of velocity, acceleration, and jerk (time derivative of acceleration). Dynamic limit is the maximum torque that the used motor can provide [9].

For our robot, its dynamic model is not yet complete, so it is impossible to predict the necessary torques for the desired TCP movement. Therefore only the kinematic limits are considered in the rest of this thesis. In addition, there can be some kinematic limits defined for the TCP of the robot as well. The motion law is used not to exceed these limits simply by modifying the velocities and accelerations along the path [9]. By applying the time scaling to the path, the parametric curve gets reparametrized. That means that the individual points for some specific time instances change, but the geometrical shape of the path stays the same.

Proper time scaling can be achieved either by using the limits as inputs to the motion law or by planning the trajectory based on some other input. In the latter case, the trajectory must be verified afterward to see if the limits are not exceeded (if so, some modification has to be done) [9].

Another critical issue in trajectory planning is exciting the resonant mode of the structure. This is, of course, undesirable for a pick-and-place robot, and therefore smooth trajectories have to be generated [7].

The trajectory must be continuous geometrically and parametrically up to a degree k, denoted G^k and C^k, respectively [10]. A continuity up to a degree $k = 2$ is essential to achieve our desired smoothness. The used curves are always C^2 continuous; thus, the only place where it needs to be checked is the merging points of adjacent curves. The geometric continuity of a curve is dependent only on its geometrical properties. If the two curves touch at the merging point, it is a geometric continuity G^0. If they also have a common tangent at that point, they are G^1 continuous. If they, in addition, have the common curvature at that point, they are G^2 continuous [10].

The parametric continuity also takes into account the used motion law. If the two curves have a common time derivative of the position with the used parameter (velocity vector), they are C^1 continuous. If they also share the same second time derivative of position, they are C^2 continuous [10]. The parametric continuity automatically implies the geometric continuity of the same degree [10].

If only line segments were used in the generator, the trajectory would not have even continuous velocity. To smoothen it, blending functions are used. In the existing solution for our robot, arcs (parts of a circle) are used for this purpose, making it C^1 continuous with a well-chosen parameter. In this thesis, a C^2 continuous blend is proposed. This should, in theory, reduce the vibrations of the robot that occur with the existing solution.

To sum up, the combination of geometrical path $\mathbf{X}(s)$ and time scaling function $s(t)$ is called trajectory $\mathbf{X}(s(t)) = \mathbf{X}(t)$ [2]. The usual trajectory generation process is to create the geometrical path in operational space based on the points defined by the user, then apply some motion law to get the time sequence of the poses of the end-effector, respecting the kinematic limits of the system and yielding a smooth trajectory [7]. Then, inverse kinematics is used to transform the generated trajectory from operational space to the joint space [7].

In this thesis, the planning process is done in advance for the whole trajectory, and after that, it is loaded to the PLC of the robot and performed. All the necessary components of the generator, described in the following Sections 3.1, 3.2, and 3.3,

were implemented in a set of Matlab functions. The PLC runs with a constant time cycle; therefore, the trajectory must be generated for discrete time instants. This brings additional problems, such as overruns, described in Section 3.4. The algorithm of the generator, which uses all the components, is described in Section 3.5. A user interface for easier use of the generator is shown in Section 3.6.

3.1 Types of used path segments

In this part, path segments that are used in the generator are described. Four different types of path segments are defined based on the most common movements for a pick-and-place robot

- Line
- Arc
- Blend
- Reorientation of the tool

Lines and arcs are described in the following paragraphs. A blend is a curve that connects two intersecting lines with an angle between them so that the path is smooth and the total length is shorter. A derivation of a curve that satisfies these constraints, as well as a more detailed description of its properties, is shown in Section 3.3.

Reorientation of the tool is an operation in which only the orientation of the TCP coordinate frame changes and the position stays the same. Because of that, the path of this segment degenerates into a point.

3.1.1 Line

A line between points $\mathbf{A}$ and $\mathbf{B}$ can be defined in parametric form by its direction vector

$$\mathbf{n} = \frac{\mathbf{B} - \mathbf{A}}{||\mathbf{B} - \mathbf{A}||} \tag{3.1}$$

as

$$\mathbf{X}(s) = \mathbf{A} + s \cdot \mathbf{n}, \tag{3.2}$$

where s is the parameter. Length of this line is

$$\ell_{\mathrm{AB}} = ||\mathbf{A} - \mathbf{B}||, \tag{3.3}$$

and therefore, to span the whole line segment

$$s \in \langle 0, \ell_{\mathrm{AB}} \rangle .$$

Velocity and acceleration on this line can be expressed by differentiating equation (3.2) with respect to time:

$$\dot{\mathbf{X}}(\dot{s}) = \dot{s} \cdot \mathbf{n} \tag{3.4}$$

$$\ddot{\mathbf{X}}(\ddot{s}) = \ddot{s} \cdot \mathbf{n} \tag{3.5}$$

3.1.2 Arc

An arc is a part of a circumference of a circle. The points of the arc in space can be calculated in many ways. One of them is a parametric form

$$\mathbf{X}(\theta) = \mathbf{C} + r \cdot \cos(\theta) \cdot \mathbf{a} + r \cdot \sin(\theta) \cdot \mathbf{b}, \tag{3.6}$$

where the parameter θ is a circle's sector angle, r is the radius of the circle, $\mathbf{C}$ is the center of the circle, and $\mathbf{a}$, $\mathbf{b}$ are vectors responsible for the rotation of the circle in space. By changing the interval of numbers for angle θ, one can choose a specific arc section of the circle. Also, an arc-length parameter s can be used instead of the parameter θ by simple substitution

$$\theta = \frac{s}{r}. \tag{3.7}$$

For this application, the most convenient way to specify the arc/circle is by three points $\mathbf{D}, \mathbf{E}, \mathbf{F}$ laying on the circle. Then an arc starting at the point $\mathbf{D}$, going through the point $\mathbf{E}$ and ending at the point $\mathbf{F}$ can be computed as suggested in [11].

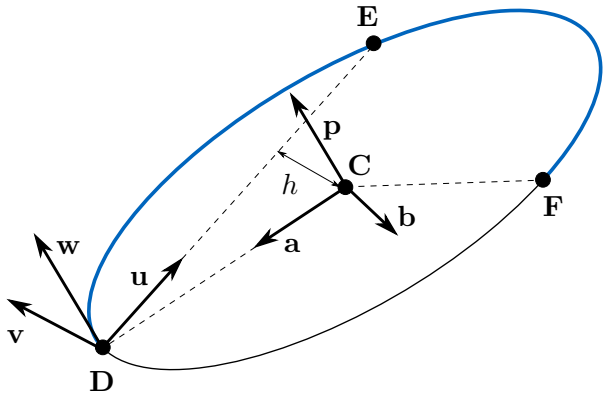


Figure 3.1: Circular path in space defined by three points

First, an orthonormal coordinate system $\mathbf{u}, \mathbf{v}, \mathbf{w}$ is created such that the origin is at the point $\mathbf{D}$, the vector $\mathbf{u}$ points towards the point $\mathbf{E}$ and the vectors $\mathbf{u}$, $\mathbf{v}$ span the plane in which the three points lay (see Fig. 3.1), by the following equations

$$\begin{aligned} \mathbf{u} &= \frac{\mathbf{u_1}}{||\mathbf{u_1}||}, \\ \mathbf{w} &= \frac{\mathbf{w_1}}{||\mathbf{w_1}||}, \\ \mathbf{v} &= \mathbf{w} \times \mathbf{u}, \end{aligned} \tag{3.8}$$

where

$$\begin{aligned} \mathbf{u_1} &= \mathbf{E} - \mathbf{D}, \\ \mathbf{w_1} &= (\mathbf{F} - \mathbf{D}) \times \mathbf{u}. \end{aligned} \tag{3.9}$$

After that, the center of the circle can be expressed using that orthonormal coordinate system as

$$\mathbf{C} = \mathbf{D} + \frac{e_x}{2} \cdot \mathbf{u} + h \cdot \mathbf{v},$$

where

$$\begin{aligned} e_x &= ||\mathbf{E} - \mathbf{D}||, \\ f_x &= (\mathbf{F} - \mathbf{D}) \cdot \mathbf{u}, \\ f_y &= (\mathbf{F} - \mathbf{D}) \cdot \mathbf{v}, \\ h &= \frac{(f_x - e_x/2)^2 + f_y^2 - (e_x/2)^2}{2 f_y}. \end{aligned} \tag{3.10}$$

The radius of the circle is

$$r = ||\mathbf{C} - \mathbf{D}|| \tag{3.11}$$

and the two vectors for rotation of the circle in space can be calculated as

$$\mathbf{a} = \mathbf{D} - \mathbf{C}, \tag{3.12a}$$

$$\mathbf{b} = \mathbf{w} \times \mathbf{a}. \tag{3.12b}$$

Then the parameter θ in (3.6) is either for a full circle $\theta \in \langle 0, 2\pi)$, or for an arc segment $\theta \in \langle 0, \theta_{\max} \rangle$

$$\theta_{\max} = \begin{cases} \arccos\left(\dfrac{(\mathbf{D} - \mathbf{C}) \cdot (\mathbf{F} - \mathbf{C})}{||\mathbf{D} - \mathbf{C}|| \cdot ||\mathbf{F} - \mathbf{C}||}\right), & \text{for } \mathbf{p} \uparrow\downarrow \mathbf{w} \\ 2\pi - \arccos\left(\dfrac{(\mathbf{D} - \mathbf{C}) \cdot (\mathbf{F} - \mathbf{C})}{||\mathbf{D} - \mathbf{C}|| \cdot ||\mathbf{F} - \mathbf{C}||}\right), & \text{for } \mathbf{p} \uparrow\uparrow \mathbf{w}, \end{cases} \tag{3.13}$$

where

$$\mathbf{p} = (\mathbf{D} - \mathbf{C}) \times (\mathbf{F} - \mathbf{C}). \tag{3.14}$$

Special cases

The case where the points $\mathbf{D}$ and $\mathbf{F}$ coincide has to be handled separately because $||\mathbf{w_1}|| = 0$ and therefore $\mathbf{w}$ in equation (3.8) cannot be computed. In other words, now there are only two points with different coordinates, and there is no unique solution for the circle defined by these points. The chosen approach assumes that the point $\mathbf{E}$ lies directly on the opposite side from the point $\mathbf{D} = \mathbf{F}$ and therefore, the center of the circle is

$$\mathbf{C} = \frac{\mathbf{D} + \mathbf{E}}{2}. \tag{3.15}$$

Radius r can be calculated as in the first case, using equation (3.11), and since the first and last points of the arc coincide, a full circle is expected, thus $\theta \in \langle 0, 2\pi)$.

The second assumption is that the vector $\mathbf{b}$ is horizontal; therefore, the z component of the normal vector $\mathbf{w}$ of the circle (to keep the notation as in the first case) is largest of all the possible rotations of the circle in space. The vector $\mathbf{a} = \left[a_x, a_y, a_z\right]^{\mathrm{T}}$ can be calculated as in (3.12). Then, the normal vector $\mathbf{w}$ can be expressed as

$$\mathbf{w_1} = \begin{cases} \left[a_x, a_y, \dfrac{-a_x^2 - a_y^2}{a_z}\right]^{\mathrm{T}}, & \text{for } a_z < 0 \\ \left[0, 0, 1\right]^{\mathrm{T}}, & \text{for } a_z = 0 \\ \left[-a_x, -a_y, \dfrac{a_x^2 + a_y^2}{a_z}\right]^{\mathrm{T}}, & \text{for } a_z > 0 \end{cases} \tag{3.16}$$

$$\mathbf{w} = \frac{\mathbf{w_1}}{||\mathbf{w_1}||}$$

The elements of the vector $\mathbf{w_1}$ are obtained by choosing the first two elements according to the geometry and then computing the third element from the constraint $\mathbf{w_1} \cdot \mathbf{a} = 0$ because the vectors must be perpendicular. After that, the vector $\mathbf{b}$ can be calculated as in (3.12b). Finally, all of these variables can be used in equation (3.6) to obtain the points $\mathbf{X}(\theta)$ that make the arc.

The cases where either $\mathbf{D} = \mathbf{E}$ or $\mathbf{E} = \mathbf{F}$ are forbidden in the generator since they are meaningless.

Velocity and acceleration

After getting the points of the arc $\mathbf{X}(\theta)$, one can get the velocity $\dot{\mathbf{X}}(\theta)$ and the acceleration $\ddot{\mathbf{X}}(\theta)$ at each point by differentiation of $\mathbf{X}(\theta)$ in equation (3.6) with respect to time:

$$\dot{\mathbf{X}}(\theta, \dot{\theta}) = -r \cdot \sin(\theta) \cdot \dot{\theta} \cdot \mathbf{a} + r \cdot \cos(\theta) \cdot \dot{\theta} \cdot \mathbf{b} \tag{3.17}$$

$$\ddot{\mathbf{X}}(\theta, \dot{\theta}, \ddot{\theta}) = -r \cdot \cos(\theta) \cdot \dot{\theta}^2 \cdot \mathbf{a} - r \cdot \sin(\theta) \cdot \ddot{\theta} \cdot \mathbf{a} - r \cdot \sin(\theta) \cdot \dot{\theta}^2 \cdot \mathbf{b} + r \cdot \cos(\theta) \cdot \ddot{\theta} \cdot \mathbf{b} \tag{3.18}$$

As well as in equation (3.6), one can use an arc-length parameter s instead of parameter θ in (3.17) and (3.18) with substitutions

$$\theta = \frac{s}{r}, \quad \dot{\theta} = \frac{\dot{s}}{r}, \quad \ddot{\theta} = \frac{\ddot{s}}{r}. \tag{3.19}$$

3.2 Types of used time scaling functions

Two different types of time scaling functions, also called motion laws, are used in the generator: S-curve velocity profile and quintic polynomial.

For arcs and straight lines, the S-curve is used. The reason is that the trajectories for pick-and-place operations consist mainly of straight lines (since it is the shortest path between two points), and minimizing the time period of these segments in the pick-and-place operation is crucial. Therefore, a motion law that ensures fast motion and low cycle time of operations is necessary [1].

On the other hand, discontinuous acceleration and thus varying inertial forces generate vibrations; hence smooth time scaling profile has to be chosen [9]. An S-curve velocity profile is a compromise of these requirements. It is smooth enough to be used and still produces a fast movement because the maximum velocity is being kept as long as possible. In addition to that, it is easy to implement.

For the blend segments, a quintic polynomial is used to slow down during the turn, reducing the centripetal acceleration and therefore reducing the forces acting on the end-effector of the robot. Compared to the S-curve, the shape of the quintic polynomial motion law for velocity (see the $\dot{s}$ profile in Fig. 3.3) matches the curvature of the blend (see Fig. 3.5) better. If the S-curve were used, the TCP would slow down at the beginning of the blend and then continue through the blend with a lower velocity, resulting in a longer total time period of this blend segment. The use of the quintic polynomial was inspired by the motion law used in [12].

3.2.1 S-curve velocity profile

This motion law consists of seven phases [2] (see Fig. 3.2)

1. constant positive jerk $J_{\max}$ until maximum acceleration $a_{\max}$ is reached
2. constant acceleration $a_{\max}$
3. constant negative jerk $J_{\min}$ until acceleration reaches zero and velocity reaches $v_{\max}$ (at the same time)
4. constant velocity $v_{\max}$
5. constant negative jerk $J_{\min}$ until minimum acceleration $a_{\min}$ is reached
6. constant deceleration $a_{\min}$
7. constant positive jerk $J_{\max}$ until acceleration reaches zero, velocity reaches value v_1 and position s_1 (at the same time)

Phases 1 – 3 can be grouped into a single acceleration phase, and phases 5 – 7 into a single deceleration phase, as is done in [9].

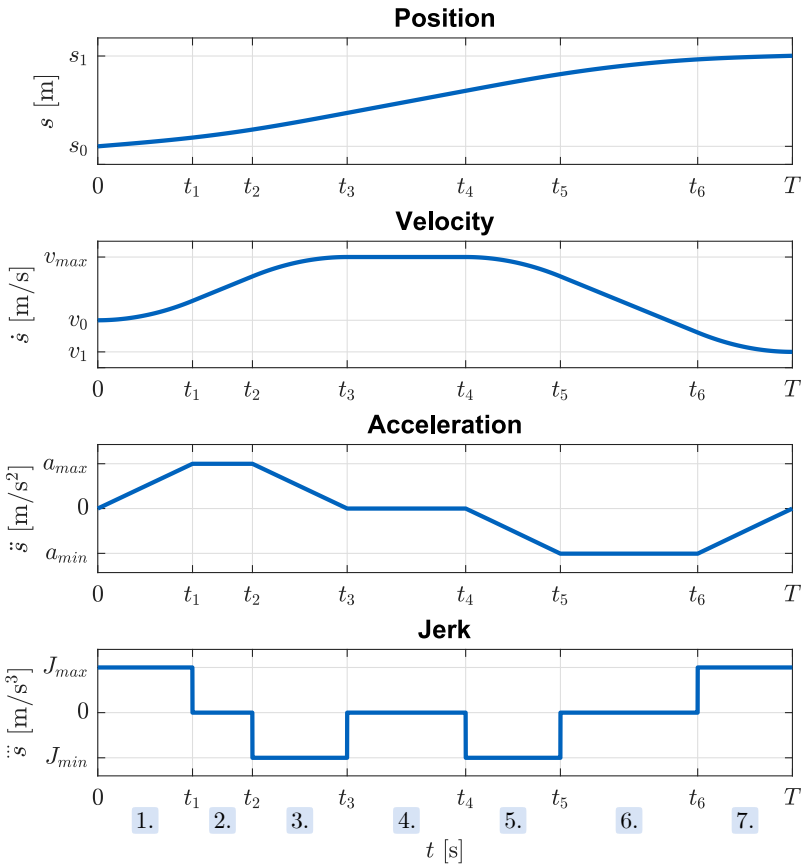


Figure 3.2: S-curve motion law

In this trajectory generator, symmetric constraints for maximum acceleration and jerk are assumed:

$$a_{\min} = -a_{\max}, \quad J_{\min} = -J_{\max}. \tag{3.20}$$

Another assumption is that

$$s_1 > s_0. \tag{3.21}$$

Given the initial and final values for position and velocity s_0, s_1, v_0 and v_1, there are two possibilities on how to define this motion law [9].

- Define $v_{\max}$, $a_{\max}$ and $J_{\max}$ using the kinematic limits. It could be either TCP kinematic limits if the trajectory in operational space is being planned or joint kinematic limits if the trajectory is planned in joint space. The task is to find the time periods for each phase and then compute the position s, velocity $\dot{s}$, acceleration $\ddot{s}$ and jerk $\dddot{s}$ according to equations for individual phases in [9].
- Define the time period for each phase. Note that symmetric limits are assumed; therefore, the duration of phases with opposite values of constants (phases 1 and 3 or phases 5 and 7) must be the same. For simplicity, only motion laws where $v_0 = v_1 = 0$ are considered in this case. Then it is sufficient

to define only the duration of constant nonzero jerk phase t_1, the duration of the grouped acceleration phase t_3, and the total time period T. The task in this case is to first compute the maximum values $v_{\max}$, $a_{\max}$ and $J_{\max}$ using equation (3.22), then continue as in the first case and at the end check if none of the kinematic limits is exceeded.

To transform the time periods into the maximum kinematic values, the following equations, derived in [9], can be used:

$$\begin{aligned} v_{\max} &= \frac{s_1 - s_0}{T - t_3} \\ a_{\max} &= \frac{s_1 - s_0}{(T - t_3)(t_3 - t_1)} \\ J_{\max} &= \frac{s_1 - s_0}{(T - t_3)(t_3 - t_1)t_1} \end{aligned} \tag{3.22}$$

There are several combinations of parameters where the trajectory cannot be realized. Therefore, a verification of the feasibility of the trajectory is necessary before computing the trajectory. One of the cases is that the total displacement $s_1 - s_0$ is so small that the velocity cannot change from v_0 to v_1 with the given acceleration and jerk limits. Another case is where the acceleration does not reach its maximum value, and thus phases with zero jerk are omitted. Also, the maximum reached velocity may be lower than the velocity limit, and therefore, a phase 4 with constant velocity is omitted. All of these cases, as well as an algorithm for this verification, and the necessary equations to calculate the time periods of the phases for every case, are shown in [9]. In the attached Matlab function for the generation of the S-curve, all the checks in [9] were implemented.

Once the time periods are known, one can compute the position s and its time derivatives $\dot{s}$, $\ddot{s}$ and $\dddot{s}$ in all phases.

3.2.2 Quintic polynomial

Polynomial of degree five for the position, as a function of time t, can be expressed as

$$s(t) = a_0 + a_1 \cdot t + a_2 \cdot t^2 + a_3 \cdot t^3 + a_4 \cdot t^4 + a_5 \cdot t^5, \tag{3.23}$$

after differentiating with respect to time, the function for velocity is

$$\dot{s}(t) = a_1 + 2 \cdot a_2 \cdot t + 3 \cdot a_3 \cdot t^2 + 4 \cdot a_4 \cdot t^3 + 5 \cdot a_5 \cdot t^4, \tag{3.24}$$

and for acceleration

$$\ddot{s}(t) = 2 \cdot a_2 + 6 \cdot a_3 \cdot t + 12 \cdot a_4 \cdot t^2 + 20 \cdot a_5 \cdot t^3. \tag{3.25}$$

The time parameter $t \in \langle 0, T \rangle$, where T is the total time period of this segment.

There are six coefficients a_0, a_1, a_2, a_3, a_4 and a_5, therefore, six boundary conditions are needed. The position at the beginning will be denoted s_0 (in general nonzero, so that it can be used after another segment and also to be able to compensate overruns, discussed in Section 3.4). The length of the path will

be denoted ℓ_{blend}. The velocity at the beginning and the end is v_0. The values s_0 and v_0 will be set to match the final values of the previous segment, and the length ℓ_{blend} is given from the geometry of the blend, discussed in Section 3.3. Finally, the acceleration at the beginning and the end is zero to match it with the S-curve profile, which will be in the previous and next segments. These conditions can also be seen in Fig. 3.3.

Therefore, the six constraints are as follows:

$$\begin{aligned} s(0) &= s_0 & s(T) &= s_1 = s_0 + \ell_{\text{blend}} \\ \dot{s}(0) &= v_0 & \dot{s}(T) &= v_0 \\ \ddot{s}(0) &= 0 & \ddot{s}(T) &= 0 \end{aligned} \tag{3.26}$$

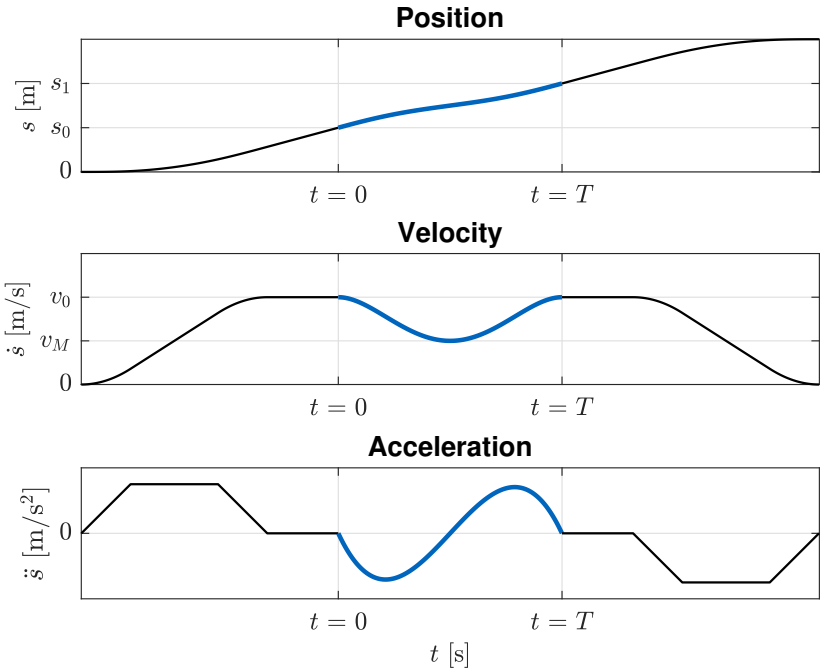


Figure 3.3: Quintic polynomial motion law (blue curve) connected to S-curve motion laws of adjacent segments

When combining constraints (3.26) with (3.23), (3.24) and (3.25), one can get the formulas for coefficients

$$\begin{aligned} a_0 &= s_0, \\ a_1 &= v_0, \\ a_2 &= 0, \\ a_3 &= \frac{10(\ell_{\text{blend}} - T \cdot v_0)}{T^3}, \\ a_4 &= \frac{15(-\ell_{\text{blend}} + T \cdot v_0)}{T^4}, \\ a_5 &= \frac{6(\ell_{\text{blend}} - T \cdot v_0)}{T^5}. \end{aligned} \tag{3.27}$$

Since it is more user-friendly to specify the velocity in the middle of the segment v_M (lowest velocity) instead of the total time period T, one more constraint is defined:

$$\dot{s}\left(\frac{T}{2}\right) = v_M \tag{3.28}$$

Combining it with equation (3.24) and with the coefficients in (3.27), one can get

$$\begin{aligned} v_M =& v_0 + 3 \cdot \frac{10(\ell_{\text{blend}} - T \cdot v_0)}{T^3} \cdot \left(\frac{T}{2}\right)^2 + \\ &+ 4 \cdot \frac{15(-\ell_{\text{blend}} + T \cdot v_0)}{T^4} \cdot \left(\frac{T}{2}\right)^3 + 5 \cdot \frac{6(\ell_{\text{blend}} - T \cdot v_0)}{T^5} \cdot \left(\frac{T}{2}\right)^4 \end{aligned} \tag{3.29}$$

and after simplification

$$v_M = \frac{15 \cdot \ell_{\text{blend}} - 7 \cdot T \cdot v_0}{8 \cdot T}. \tag{3.30}$$

The last step is to solve for T

$$T = \frac{15 \cdot \ell_{\text{blend}}}{7 \cdot v_0 + 8 \cdot v_M}. \tag{3.31}$$

It may be more convenient not to specify the velocity v_M directly, but rather in a relative value, how much to slow down compared to the initial velocity v_0

$$p = \frac{v_0 - v_M}{v_0}. \tag{3.32}$$

Thus, the total time period is

$$T = \frac{15 \cdot \ell_{\text{blend}}}{15 \cdot v_0 - 8 \cdot p \cdot v_0}. \tag{3.33}$$

3.3 Blends

Inspired by the work in [13] and [12], a quintic Pythagorean-Hodograph (PH) curve is chosen to create the blend between lines. However, the proposed method considers only right-angled corners, which may be insufficient for our purposes. Our robot has 5 DoF, and because of that, the pick and place operations do not have to be vertical, as it is with standard 3-DoF Delta robots. This means that there could be any angle between the lines of the pick-and-place path. Therefore, in this thesis, the approach of the authors in [13] and [12] is extended to a solution for creating a blending curve between two lines with an arbitrary angle between them.

3.3.1 Properties of Pythagorean-Hodograph curves

Pythagorean-Hodograph curve is a type of Bezier curve [13]. In this thesis, a PH quintic is considered, as in [13], [12] and [14] because of the limited shape flexibility of cubic PH curves [14]. Also, only planar curves are considered, because two intersecting lines always lie in one plane, so a spatial blend will never be necessary.

A planar quintic PH curve can be expressed as

$$\mathbf{X}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \sum_{i=0}^{5} \mathbf{P}_i \cdot t^i \cdot (1-t)^{5-i}, \tag{3.34}$$

where $t \in \langle 0, 1 \rangle$ and $\mathbf{P}_i$ are the control points shown in equations (3.38).

For a Pythagorean-Hodograph curve, a polynomial function $\sigma(t)$ that satisfies the following condition for every t must exist [14]:

$$x'^2(t) + y'^2(t) = \sigma^2(t) \tag{3.35}$$

This can be achieved, as shown in [13] and [14], by defining

$$\begin{aligned} x'(t) &= u^2(t) - v^2(t), \\ y'(t) &= 2 \cdot u(t) \cdot v(t), \end{aligned} \tag{3.36}$$

where

$$\begin{aligned} u(t) &= u_0(1-t)^2 + 2u_1(1-t)t + u_2t^2, \\ v(t) &= v_0(1-t)^2 + 2v_1(1-t)t + v_2t^2. \end{aligned} \tag{3.37}$$

After combining equation (3.34) with integrated equation (3.36) and (3.37),

the control points $\mathbf{P}_i$ in (3.34) can be expressed as [13]:

$$\mathbf{P}_1 = \mathbf{P}_0 + \frac{1}{5}\begin{bmatrix} 2u_0v_0 \\ u_0^2 - v_0^2 \end{bmatrix} \tag{3.38a}$$

$$\mathbf{P}_2 = \mathbf{P}_1 + \frac{1}{5}\begin{bmatrix} u_0v_1 + u_1v_0 \\ u_0u_1 - v_0v_1 \end{bmatrix} \tag{3.38b}$$

$$\mathbf{P}_3 = \mathbf{P}_2 + \frac{2}{15}\begin{bmatrix} 2u_1v_1 \\ u_1^2 - v_1^2 \end{bmatrix} + \frac{1}{15}\begin{bmatrix} u_0v_2 + u_2v_0 \\ u_0u_2 - v_0v_2 \end{bmatrix} \tag{3.38c}$$

$$\mathbf{P}_4 = \mathbf{P}_3 + \frac{1}{5}\begin{bmatrix} u_1v_2 + u_2v_1 \\ u_1u_2 - v_1v_2 \end{bmatrix} \tag{3.38d}$$

$$\mathbf{P}_5 = \mathbf{P}_4 + \frac{1}{5}\begin{bmatrix} 2u_2v_2 \\ u_2^2 - v_2^2 \end{bmatrix} \tag{3.38e}$$

The main advantage of PH curves for our purpose is that an arc length of this curve can be expressed as a polynomial function of the curve parameter. In addition, the arc length is monotonically increasing. This implies that a task to find a parameter t in (3.34) (and therefore find a point on the curve) with a given arc length can be computed with relatively low computational cost. In our case, where a quintic PH curve is used, it is a polynomial of degree five, which can be solved numerically. Arc length for an arbitrary point, where $t = t_1$, can be computed as

$$s(t_1) = \int_0^{t_1} \sigma(t)\,dt = \int_0^{t_1} \sqrt{x'^2(t) + y'^2(t)}\,dt. \tag{3.39}$$

When the physical limits of the robot are taken into account, only a few iterations of a Newton-Raphson iteration method are necessary to find a solution within the robot's accuracy [15]. Finding a parameter t for some given arc length is necessary for reparametrization of the curve (see Section 3.3.3), and thanks to this property of PH curves, it can be done in real-time [12]. Thus a trajectory can be generated during the movement of the robot.

3.3.2 Derivation of the curve for blends

To simplify the derivation, the curve is always considered to be in the xy plane, with the start point $\mathbf{P}_0$ being the origin of the coordinate system and the first line being vertical (coinciding with the y axis), as shown in Fig. 3.4. Also, the curve always creates a right-handed turn. In other words, the x component of the point $\mathbf{P}_5$ is positive. To use it for any arbitrary connection of two lines in space, this curve has to be rotated and translated appropriately. This is described in detail in Section 3.3.5.

The geometry of the curve is given by two parameters: angle θ between the lines that are supposed to merge and radius ϱ, which controls the size of the blend (how far away from the intersection of the lines should the blend start).

Since the start of the curve is at the origin, the first line is vertical, and the radius is ϱ, the intersection of the lines lies at $[0, \varrho]^{\mathrm{T}}$ and the second line is at

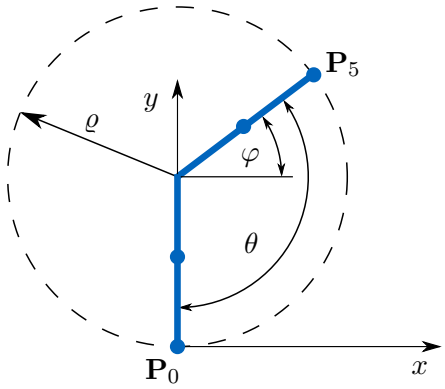


Figure 3.4: Geometry of control points of the blending curve

angle φ with positive x axis, where

$$\varphi = \theta - \frac{\pi}{2}. \tag{3.40}$$

In general, in equation (3.38) there are six parameters u_0, u_1, u_2, v_0, v_1, and v_2, which control the quintic PH curve. To reduce the number of unknowns, let us make some assumptions, as in [13]. The line between $\mathbf{P}_0$ and $\mathbf{P}_1$ is vertical, so $v_0 = 0$. Since we blend two straight line segments, the curvature at both ends of the curve $\kappa(0) = \kappa(1) = 0$. Therefore, $v_1 = u_1 = 0$.

After that, general equations (3.38) are simplified into:

$$\mathbf{P}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{3.41a}$$

$$\mathbf{P}_1 = \mathbf{P}_2 = \begin{bmatrix} 0 \\ \frac{u_0^2}{5} \end{bmatrix} \tag{3.41b}$$

$$\mathbf{P}_3 = \mathbf{P}_4 = \begin{bmatrix} \frac{u_0 v_2}{15} \\ \frac{u_0^2}{5} + \frac{u_0 u_2}{15} \end{bmatrix} \tag{3.41c}$$

$$\mathbf{P}_5 = \begin{bmatrix} \frac{u_0 v_2}{15} + \frac{2u_2 v_2}{5} \\ \frac{u_0^2}{5} + \frac{u_0 u_2}{15} + \frac{u_2^2}{5} - \frac{v_2^2}{5} \end{bmatrix} \tag{3.41d}$$

Now, we are left with only three unknowns u_0, u_2 and v_2. We will find them from the following equations.

The first two equations can be derived from the geometric constraints of the Cartesian coordinates of the point $\mathbf{P}_5$ (as shown in Fig. 3.4):

$$\mathbf{P}_5 = \begin{bmatrix} P_{5x} \\ P_{5y} \end{bmatrix} = \begin{bmatrix} \varrho \cdot \cos(\varphi) \\ \varrho \cdot (1 + \sin(\varphi)) \end{bmatrix} \tag{3.42}$$

Combining equations (3.41d) and (3.42), we get

$$\frac{u_0 v_2}{15} + \frac{2u_2 v_2}{5} = \varrho \cdot \cos(\varphi) \tag{3.43}$$

and

$$\frac{u_0^2}{5} + \frac{u_0 u_2}{15} + \frac{u_2^2}{5} - \frac{v_2^2}{5} = \varrho \cdot (1 + \sin(\varphi)). \tag{3.44}$$

Another condition for obtaining the desired path is that we require a symmetric curvature profile of the curve and, therefore, extrema in the middle, thus

$$\kappa'\left(\frac{1}{2}\right) = 0. \tag{3.45}$$

The curvature of this PH curve (see Fig. 3.5), according to [13], can be expressed as

$$\kappa(t) = \frac{4u_0 v_2 (t - t^2)}{(u^2(t) + v^2(t))}. \tag{3.46}$$

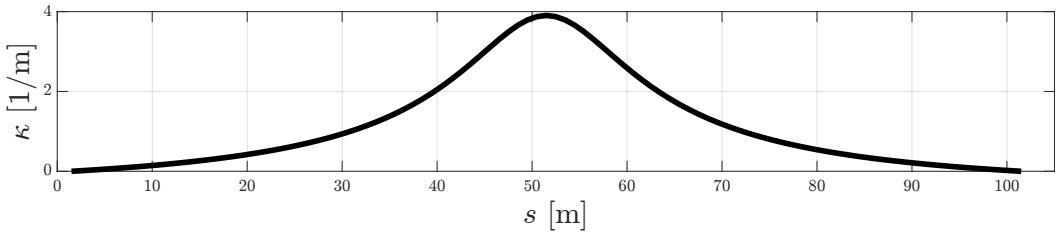


Figure 3.5: Curvature of the PH curve used for blend as a function of its arc length

Combining the time derivative of (3.46) with (3.45), the following formula for $t = \frac{1}{2}$ can be written

$$\frac{-4 \cdot u_0 \cdot v_2 (v\left(\frac{1}{2}\right) \cdot v'\left(\frac{1}{2}\right) + u\left(\frac{1}{2}\right) \cdot u'\left(\frac{1}{2}\right)) \cdot \left(v^2\left(\frac{1}{2}\right) + u^2\left(\frac{1}{2}\right)\right)}{\left(u^2\left(\frac{1}{2}\right) + v^2\left(\frac{1}{2}\right)\right)^4} = 0, \tag{3.47}$$

where, according to [13],

$$\begin{aligned} u(t) &= u_0 \cdot (1-t)^2 + u_2 \cdot t^2, \\ u'(t) &= -2 \cdot u_0 + 2 \cdot u_0 \cdot t + 2 \cdot u_2 \cdot t, \\ v(t) &= v_2 \cdot t^2, \\ v'(t) &= 2 \cdot v_2 \cdot t, \end{aligned} \tag{3.48}$$

and for $t = \frac{1}{2}$

$$\begin{aligned} u\left(\frac{1}{2}\right) &= u_0 \cdot \left(1 - \frac{1}{2}\right)^2 + u_2 \cdot \left(\frac{1}{2}\right)^2 = \frac{u_0 + u_2}{4}, \\ u'\left(\frac{1}{2}\right) &= -2 \cdot u_0 + 2 \cdot u_0 \cdot \frac{1}{2} + 2 \cdot u_2 \cdot \frac{1}{2} = -u_0 + u_2, \\ v\left(\frac{1}{2}\right) &= v_2 \cdot \left(\frac{1}{2}\right)^2 = \frac{v_2}{4}, \\ v'\left(\frac{1}{2}\right) &= 2 \cdot v_2 \cdot \frac{1}{2} = v_2. \end{aligned} \tag{3.49}$$

Now, equations (3.43), (3.44), and (3.47), together with (3.49), can be solved to get u_0, u_2 and v_2. Since these equations are fairly complicated to solve in hand, Symbolic Math Toolbox in Matlab by MathWorks [16] was used. The script for finding these solutions is called `derivation_of_equations_for_blending`.

It gave us eight solutions (sets of formulas for u_0, u_2 and v_2). To choose the correct one, the curves for the eight solutions were plotted in an example for $\varrho = 1$ and $\varphi = 10°$. The results are shown in Fig. 3.6. In solutions 5 - 8 (in the second row of the figure), all three unknowns u_0, u_2 and v_2 have opposite signs compared to solutions 1 - 4 (in the first row). Since in equations (3.41b) - (3.41d) they are always either in second power or multiplied by each other in pairs, they yield the same result.

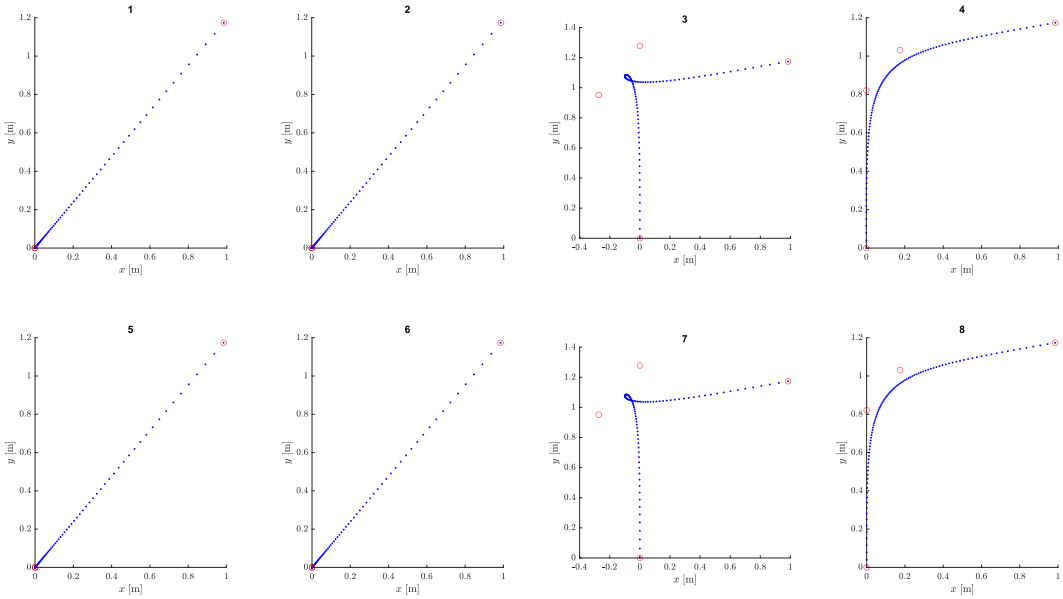


Figure 3.6: Eight different solutions for u_0, u_2 and v_2 shown in an example for $\varrho = 1$ and $\varphi = 10°$

One can see that curves in solutions 1, 2, 5, and 6 degenerate into a straight line, although still respecting the constraints about the point $\mathbf{P}_5$ and curvature in the middle. Solutions 3 and 7 also respect those constraints but create a loop, which is undesirable. Thus, solution 4 was chosen (solution 8 could also have been

chosen, as it gives the same result). The formulas found are as follows:

$$u_0 = \sec(\varphi) \cdot \sqrt{15 \cdot (1 + \sin(\varphi)) \cdot \left(\frac{\varrho \cdot (-1 + \sin(\varphi)) \left(-6 - 6\sin(\varphi) + \sqrt{2}\sqrt{1 + \sin(\varphi)} \right)}{17 + 18 \cdot \sin(\varphi)} \right)}$$

$$u_2 = \sec(\varphi) \cdot (1 + \sin(\varphi)) \cdot \sqrt{15 \cdot \frac{\varrho \cdot \cos^2(\varphi) \left(-6 - 6\sin(\varphi) + \sqrt{2}\sqrt{1 + \sin(\varphi)} \right)}{\cos^2(\varphi) - 35 \cdot (1 + \sin(\varphi))^2}}$$

$$v_2 = \sqrt{15 \cdot \frac{\varrho \cdot \cos^2(\varphi) \left(-6 - 6\sin(\varphi) + \sqrt{2}\sqrt{1 + \sin(\varphi)} \right)}{\cos^2(\varphi) - 35 \cdot (1 + \sin(\varphi))^2}} \tag{3.50}$$

To sum up the procedure of creating this curve, for a given angle φ and radius ϱ, values u_0, u_2 and v_2 are computed using (3.50), then inserted into (3.41) to get the control points. After that, these control points are used in (3.34) to construct the curve.

3.3.3 Reparametrization

To apply the quintic polynomial motion law, described in Section 3.2.2, the blend must be first rendered with a uniform arc length [13]. In other types of path segments, such as lines and arcs (in Section 3.1), this did not have to be considered because they are naturally with uniform arc length rendering. The effect of reparametrization can be seen in Fig. 3.7. On the left, the original curve with constant time increments Δt is plotted. On the right side, the reparametrized version for constant arc length increments Δs is plotted. Note that the shape of the curve stays the same, only the spacing of the individual points changes.

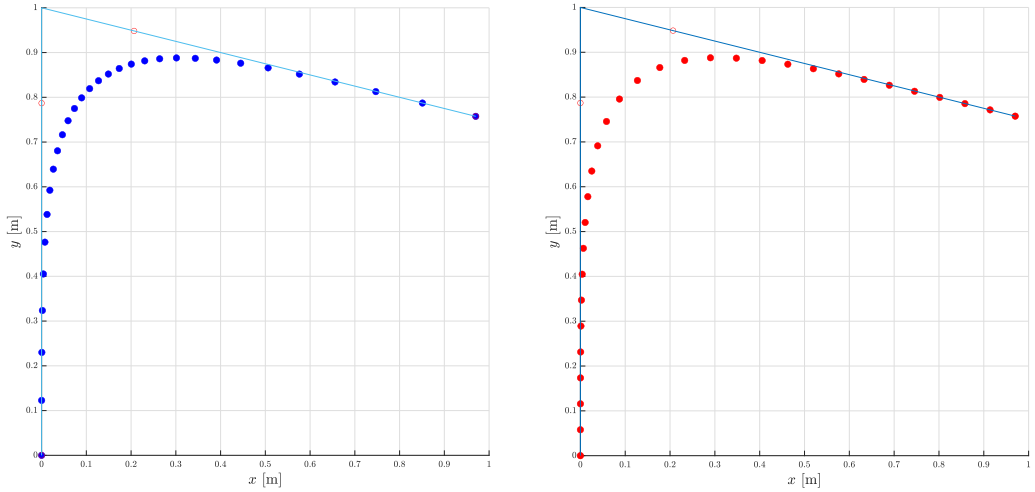


Figure 3.7: A blend using a PH curve. Natural parametrization with constant Δt (left) and reparametrized for constant arc length increments Δs (right).

The importance of reparametrization of the curve can also be seen when the continuity of the merge of the blend curve with a line is analyzed. The natural parametrization does not give continuous velocity and acceleration profiles. The velocity and acceleration vectors at the merge point have the same orientation for both curves, but their magnitudes differ. Thus, the blend with natural parametrization is only geometrically continuous but not parametrically. With the correctly reparametrized curve, even the magnitudes of the velocity and acceleration vectors are the same for both curves, and thus the trajectory is C^2 continuous.

The task is to find a parameter γ such that the arc length of the points of the blend is given by some function, the easiest being a linear function for uniform spacing. For that, a relation between the parameter γ and the arc length s is needed. Equation (3.39) together with equation (3.36) can be used for this purpose [13]:

$$\begin{aligned} s(\gamma) = \int_0^{\gamma} \sqrt{x'^2(\gamma) + y'^2(\gamma)}\, dt = (u_0^2 + 2u_0u_2 + u_2^2 + v_2^2)\frac{\gamma^5}{5} - \\ - (4u_0^2 + 4u_0u_2)\frac{\gamma^4}{4} + (6u_0^2 + 2u_0u_2)\frac{\gamma^3}{3} - 4u_0^2\frac{\gamma^2}{2} + u_0^2\gamma. \end{aligned} \tag{3.51}$$

For some given arc length s, one can find the root of the following polynomial:

$$\begin{aligned} &(u_0^2 + 2u_0u_2 + u_2^2 + v_2^2)\frac{\gamma^5}{5} - \\ &- \gamma^4 \cdot (u_0^2 + u_2 \cdot u_0) + (6u_0^2 + 2u_0u_2)\frac{\gamma^3}{3} - 2u_0^2\gamma^2 + u_0^2\gamma - s = 0 \end{aligned} \tag{3.52}$$

where the arc length s can be either given by a linear function of time, to create a curve with constant Δs as in Fig. 3.7, or given by some time scaling function $s(t)$. Since $s(\gamma)$ is a function that increases monotonically, there is only one root.

In the actual trajectory generator, the quintic polynomial motion law (Section 3.2.2) is used to get a value of s for a given time t. This value is then inserted into (3.52) and a root γ of this polynomial can be found. After that, a point on the reparametrized curve, which respects the given motion law, is created by inserting the found γ into (3.34). The control points of the curve are still given by its geometry and can be computed as in the previous Section 3.3.2. Doing this for every time instance t, the whole trajectory of the blend, with a proper time scaling, is created.

3.3.4 Velocity and acceleration

The next step is to find velocity and acceleration after reparametrization. A chain rule has to be used when differentiating it to get the velocity and acceleration of $\mathbf{X}(\gamma(s(t)))$ since a parameter of a point on the curve is a function of another parameter.

The final formulas for both velocity and acceleration are quite long, and it does not make sense to show them in here; therefore, only the process of their creation is shown, and the final equations can be found in the attached code since

if anybody needs them for their application, they will serve better as a piece of code.

Velocity

The velocity can be expressed as

$$\dot{\mathbf{X}}(\gamma) = \frac{\mathrm{d}\mathbf{X}(\gamma(s(t)))}{\mathrm{d}t} = \frac{\mathrm{d}\mathbf{X}}{\mathrm{d}\gamma}\frac{\mathrm{d}\gamma}{\mathrm{d}s}\frac{\mathrm{d}s}{\mathrm{d}t}, \tag{3.53}$$

where

$$\frac{\mathrm{d}\mathbf{X}}{\mathrm{d}\gamma} = \begin{bmatrix} \dfrac{\mathrm{d}x}{\mathrm{d}\gamma} \\ \dfrac{\mathrm{d}y}{\mathrm{d}\gamma} \end{bmatrix} \tag{3.54}$$

can be differentiated using $\mathbf{X}(\gamma)$ from (3.34) with control points (3.41) straightforwardly. The term $\dfrac{\mathrm{d}s}{\mathrm{d}t} = \dot{s}$ is already given from the motion law. Finding $\dfrac{\mathrm{d}\gamma}{\mathrm{d}s}$ requires having the formula for $\gamma(s)$, but since $s(\gamma)$ is known from equation (3.51), the rule for the derivative of the inverse function can be used:

$$\frac{\mathrm{d}\gamma}{\mathrm{d}s} = \frac{1}{\dfrac{\mathrm{d}s(\gamma)}{\mathrm{d}\gamma}} \tag{3.55}$$

The differentiated terms can be found in the attached Matlab function `blending`.

Acceleration

The acceleration can be expressed as

$$\ddot{\mathbf{X}}(\gamma) = \frac{\mathrm{d}\dot{\mathbf{X}}(\gamma)}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}\mathbf{X}}{\mathrm{d}\gamma}\frac{\mathrm{d}\gamma}{\mathrm{d}s}\frac{\mathrm{d}s}{\mathrm{d}t}\right). \tag{3.56}$$

When applying the product rule, one can get

$$\ddot{\mathbf{X}}(\gamma) = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}\mathbf{X}}{\mathrm{d}\gamma}\right)\frac{\mathrm{d}\gamma}{\mathrm{d}s}\frac{\mathrm{d}s}{\mathrm{d}t} + \frac{\mathrm{d}\mathbf{X}}{\mathrm{d}\gamma}\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}\gamma}{\mathrm{d}s}\right)\frac{\mathrm{d}s}{\mathrm{d}t} + \frac{\mathrm{d}\mathbf{X}}{\mathrm{d}\gamma}\frac{\mathrm{d}\gamma}{\mathrm{d}s}\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}s}{\mathrm{d}t}\right) \tag{3.57}$$

and after handling the time derivatives in each term, the equation for acceleration is created

$$\ddot{\mathbf{X}}(\gamma) = \frac{\mathrm{d}^2\mathbf{X}}{\mathrm{d}\gamma^2}\xi^2\dot{s}^2 + \frac{\mathrm{d}\mathbf{X}}{\mathrm{d}\gamma}\frac{\mathrm{d}\xi}{\mathrm{d}\gamma}\xi\dot{s}^2 + \frac{\mathrm{d}\mathbf{X}}{\mathrm{d}\gamma}\xi\ddot{s}, \tag{3.58}$$

where $\xi = \dfrac{\mathrm{d}\gamma}{\mathrm{d}s}$ for simplicity. All of the terms can be differentiated straightforwardly from already existing formulas or are given from the used motion law. They can also be found in the attached Matlab function `blending`.

3.3.5 Use case in a blend

In the trajectory generator, a blend is created between two line segments, see Fig. 3.8. Let the first line segment be between points **A** and **B**, and the second line segment between points **B** and **C**. Then, if there is a nonzero radius ϱ defined at point **B**, the blend is created at that point as follows. First, directional vectors

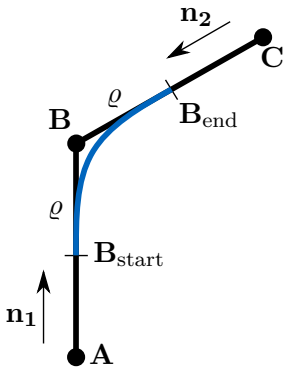


Figure 3.8: A blend between two lines

of the lines pointing towards the intersection point are computed:

$$\begin{aligned} \mathbf{n_1} &= \frac{\mathbf{B}-\mathbf{A}}{\|\mathbf{B}-\mathbf{A}\|} \\ \mathbf{n_2} &= \frac{\mathbf{B}-\mathbf{C}}{\|\mathbf{B}-\mathbf{C}\|} \end{aligned} \tag{3.59}$$

After that, the angle between the lines θ and an angle φ between the second line and x axis are

$$\theta = \arccos\left(\frac{\mathbf{n_1}\cdot\mathbf{n_2}}{\|\mathbf{n_1}\|\cdot\|\mathbf{n_2}\|}\right), \quad \varphi = \theta - \frac{\pi}{2}. \tag{3.60}$$

Points where the blend should start and end are

$$\begin{aligned} \mathbf{B}_{\text{start}} &= \mathbf{B} - \varrho\cdot\mathbf{n_1}, \\ \mathbf{B}_{\text{end}} &= \mathbf{B} - \varrho\cdot\mathbf{n_2}. \end{aligned} \tag{3.61}$$

The next step is to find the control points of a blend curve in the xy plane based on ϱ and φ. This is done using equation (3.50) to get the values u_0, u_2 and v_2 and then inserting them into (3.41) to get the control points.

The length of the blend can be found when $\gamma = 1$ is inserted into equation (3.51). This length, together with a given value of the velocity at the beginning and in the middle of the blend, can be used to generate the quintic polynomial motion law.

Note that the trajectory generator is working with discrete time instants because of the inner loop in the PLC of the robot, which sends a command with desired position and velocity to the actuators every iteration of this loop. Therefore, a time vector is created, where the values are spaced uniformly, with a

step equal to the cycle time of the PLC. For every value in this time vector, the arc length parameter s and its time derivatives are computed using (3.23) – (3.25).

Every value of s is then reparameterized, as described in Section 3.3.3, to get a corresponding value of γ. This value can be inserted as a parameter into (3.34), where the control points are already known. Now, the points $\mathbf{X}_{\text{xy}}$ of the blend, respecting the motion law, are generated, but only in the xy plane with the beginning at the origin.

The last step is therefore rotating and translating the blend so that it can merge with the rest of the trajectory. The rotation can be done using a matrix

$$\mathbf{R} = \begin{bmatrix} d_1 & n_1 & c_1 \\ d_2 & n_2 & c_2 \\ d_3 & n_3 & c_3 \end{bmatrix}, \tag{3.62}$$

where the individual columns are axes of the orthonormal coordinate system $x_2\, y_2\, z_2$ of the blend expressed in the world coordinate frame $x_1\, y_1\, z_1$, see Fig. 3.9. According to the figure, they can be defined as follows:

$$\begin{aligned} \mathbf{n_1} &= \begin{bmatrix} n_1 & n_2 & n_3 \end{bmatrix}^{\mathrm{T}} \\ \mathbf{c} &= \begin{bmatrix} c_1 & c_2 & c_3 \end{bmatrix}^{\mathrm{T}} = \frac{(-\mathbf{n_1}) \times \mathbf{n_2}}{||(-\mathbf{n_1}) \times \mathbf{n_2}||} \\ \mathbf{d} &= \begin{bmatrix} d_1 & d_2 & d_3 \end{bmatrix}^{\mathrm{T}} = \frac{\mathbf{n_1} \times \mathbf{c}}{||\mathbf{n_1} \times \mathbf{c}||}. \end{aligned} \tag{3.63}$$

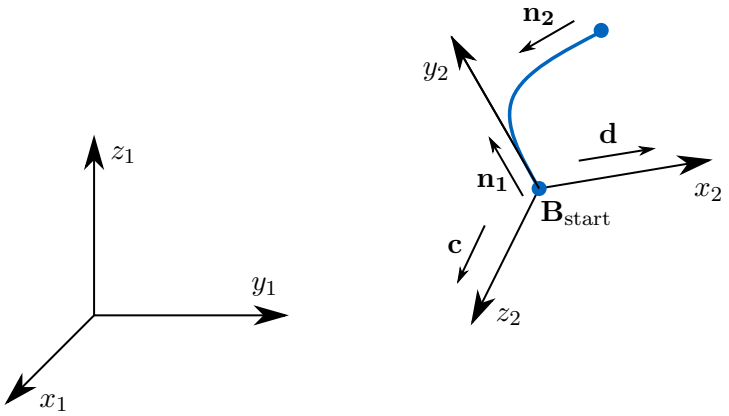


Figure 3.9: Orientation of the blend in x_2y_2 plane with respect to the world coordinate frame $x_1y_1z_1$

The curve of the blend in the proper position and with the correct orientation in space is given by rotating the points $\mathbf{X}_{\text{xy}}$ using the rotation matrix and translating it so that the first point of the blend is at the point $\mathbf{B}_{\text{start}}$. The same rotation matrix can be used to transform velocity and acceleration:

$$\begin{aligned} \mathbf{X} &= \mathbf{R} \cdot \mathbf{X}_{\text{xy}} + \mathbf{B}_{\text{start}} \\ \dot{\mathbf{X}} &= \mathbf{R} \cdot \dot{\mathbf{X}}_{\text{xy}} \\ \ddot{\mathbf{X}} &= \mathbf{R} \cdot \ddot{\mathbf{X}}_{\text{xy}} \end{aligned} \tag{3.64}$$

3.4 Overruns

To run the trajectory on the physical robot, it has to be discrete with a constant time step. Therefore, a time vector with values that increase by this constant step is created and used as input to the time scaling function. However, in most cases, the total duration of this time scaling function is not an integer multiple of the time step. Because of that, the last point of the trajectory segment is not exactly at the end of this segment but a bit closer to the starting point.

The next generated point would interfere with the following trajectory segment, which is unacceptable. Also, if the next segment started with the first point at the beginning of its segment, the two adjacent points (the last point of the first segment and the first point of the second segment) would be closer to each other than they should be. This affects the velocity and acceleration at that point – with a sudden jump in the spacing of the points, the velocity is discontinuous.

The solution is to generate one more point after each segment and measure its distance from the official end of the relevant segment. This distance is called overrun. Then, the points of the next segment are shifted by that overrun to keep the spacing as computed from the time scaling. This can be easily achieved by adding the overrun value to the starting position s_0 of the time scaling function of the next segment.

The described method can be seen in Fig. 3.10. The blue dots represent the points of the line segment, and the red dots represent the points of the blend segment. The last blue dot is just before the end of the line segment; the next one would be in the blend segment (the red dot with a blue outline). The gray dot is the transition of the two segments, where the first red dot should be, but then it would be too close to the last blue dot. Therefore, the first red dot is offset by the overrun value to be at the place where the next blue dot would be.

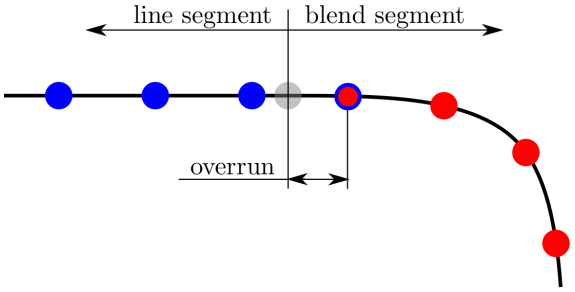


Figure 3.10: Overruns

This issue depends on the velocity at that point. When the velocity is low, or even zero, when stopping between segments, this overrun value is small enough that it may not be noticeable. However, with higher velocities, especially in a transition between line and blend segments, the gaps between the points and thus also the overrun grow, and discrepancies in velocity occur.

3.5 Algorithm of the trajectory generator

All the path and time scaling functions described above will now be combined and used in the actual trajectory generator. The generator itself is divided into two functions. The first takes the points defined by the user and creates a data structure that contains all the necessary information about the individual segments. The second function then uses this data structure to generate the trajectory in a form that is ready to be used in the physical robot. In this way, modularity of the generator is achieved since, in some applications, a different method of choosing the segments than by individual points may be better. One can easily create his method to generate the data structure and then use only the second function to get the actual trajectory.

3.5.1 Generating the segments

The first function is called `generate_segments`. As input, it takes a set of points with additional information about each one:

- coordinates of the point x, y, and z
- radius of the blend at that point ϱ
- boolean variable b if it is a second point of a triplet creating an arc
- orientation q_4, q_5 of the wrist part of the robot (axes A4 and A5) at that point

The user can create the desired path by defining these parameters at each defined point. By default, all points are connected by lines with a blend of the defined radius between them. If the blend is not wanted at some point, the radius can be set to zero. If an arc made by three points is desired instead of connecting the three points by two lines, the second point should have the boolean variable b set to logical 1. The orientation of the wrist part of the robot does not have to be defined at every point; it automatically interpolates the orientation between the two defined poses. Also, if there are two points after each other with the same coordinates but different orientations, a wrist reorientation is planned.

At the beginning of the Matlab function, the input is checked to ensure a feasible trajectory – the blend radii at the first and last points have to be zero because there is no line to blend it with. Also, the boolean variable for creating an arc b cannot be set to logical 1 for two points next to each other, and a blend cannot be planned at a point of an arc. Another condition is that the wrist orientation must be defined at the first and last point. When a reorientation is planned, there also cannot be any blend at that point.

After that, the code goes through the individual points one by one in a loop, sorts them by their type, and creates the data structure of the segments. If the point has the same coordinates as the previous one, a reorientation segment is added. If not, the boolean value b is checked, and if it is logical 1, an arc segment is added. If it is logical 0, the radius of the blend is checked. If zero, only a line

segment is planned between this and the previous point. If the blend radius is nonzero, the starting point of the blend is computed, and two segments are added – a line between the previous point and the starting point of the blend and a blend. In that case, the next iteration of this loop has to start the geometrical path of the next segment at the end of the blend and not at the point from the input.

With every segment, the necessary values are stored in the structure:

- `type` – a type of segment (`line`, `blend`, `arc` or `reorient`)
- `R` – rotation matrix, only for a blend segment, created as in (3.62) and (3.63)
- `phi` – angle between the second line and the x axis, defined in (3.40), only for a blend segment
- `rho` – radius of the blend, only for a blend segment
- `startPoint` – coordinates of the point, where this segment starts
- `middlePoint` – only for arc and blend segments, the second point from input for arc, the point form input for blend
- `endPoint` – coordinates of the point, where this segment ends
- `vector` – directional vector, only for a line segment
- `orientDefined` – boolean variable to find out if the orientation of the wrist is defined in this segment
- `q4` – angle of axis A4 (part of orientation of the wrist)
- `q5` – angle of axis A5 (part of orientation of the wrist)

In the end, all of the blend segments have to be checked to see if the blend does not reach too far and interfere with the adjacent segments. If so, the user has to change either the position of the points or the radius of the blend. The output of this function is the mentioned segment data structure.

3.5.2 Generating the trajectory

The second function is called `traj_generation`. As input, it takes the segment data structure plus the orientation of the wrist at the beginning. Its purpose is to create a set of positions q_i and velocities $\dot{q}_i$, $i = 1 \dots 5$, for actuators of the robot so that the TCP will follow the desired trajectory.

For each of the already existing segments, kinematic limits (maximum Cartesian velocity, acceleration, and jerk of TCP, as well as joint velocity, acceleration, and jerk limits) have to be assigned. This can be done either by setting global limits for all segments, as is done in this function, or by attaching the limits to the data structure of the individual segments, as in the user interface shown in Section 3.6.

In this function, there are two subsequent loops in which the individual segments are transformed into a trajectory. The first loop takes care of the time

scaling part, and the second loop combines it with the geometry and completes the trajectory generation. The reason for dividing it into two loops is that the defined orientation of the wrist part of the robot is not obligatory in every segment. However, for computing the inverse kinematics for every point of the trajectory, the orientation of the wrist at that point has to be already known. Therefore, all segments have to be run through to get the interpolated orientation of the wrist for the whole duration of the trajectory, and after that, the second loop can do the rest.

First loop

All of the segments are examined one by one to obtain the correct time scaling function and orientation of the wrist in this segment. If it is a line segment, an S-curve velocity profile is applied. The initial velocity is set according to the final velocity of the previous segment, and the final velocity is defined by the user or zero if the next segment is not a blend (and the robot has to stop).

If it is a blend segment, a function `blending` is called. It calculates the geometry of the blend based on the values `phi` and `rho` in the data structure for this segment. After that, it computes the arc length of the blend and generates a quintic polynomial motion law, with the initial and final velocity as the final velocity of the previous segment and the velocity in the middle of the blend as defined by the user. Then, it works on the reparametrization and generates the individual points, as well as velocities and accelerations, for the whole blend. These values are then compared with the velocity and acceleration limits to see if the blend is feasible.

For an arc segment, a function `circle3Dfunc` computes all the necessary variables in equation (3.6). The length of the arc $\ell = \theta_{\max} \cdot r$ is then used to generate an S-curve velocity profile. In this case, both the initial and final velocities are zero. Note that in both the blend and arc segments, the normal acceleration is not taken into account.

Finally, if it is a reorientation segment, an S-curve velocity profile is computed individually for both axes of the wrist of the robot. The lengths of the two time scaling profiles for the axes are compared, the longer one is chosen, and a new S-curve for the other axis is generated to match the duration of the phases of the chosen S-curve. The reason for doing so is that the longer duration means lower kinematic limits, and therefore slowing down the faster S-curve ensures that both axes will not exceed their limits.

In all types of segments, except for reorientation, the variable `orientDefined` is checked. If the orientation of the wrist is defined in this segment, the duration since the last defined orientation is computed, and an S-curve for both axes of the wrist are generated to match this duration.

At the end of every iteration of this loop, kinematic limits for axes A4 and A5 are checked, and all the computed variables, which will be necessary later, are added to the data structure.

Second loop

In the second loop, all segments are again examined one by one. This time, the motion law $s(t)$ is known, as well as the orientation of the wrist q_4, q_5. For line or arc segments, the parameter s is inserted into equation (3.2) or (3.6), respectively. By doing so, the points $\mathbf{X}(s)$ of the trajectory segment are created. The same is done for velocity and acceleration – equations (3.4) and (3.5) are used for a line, equations (3.17), (3.18) and (3.19) for an arc.

For blend segments, the points are already stored in the data structure, and the only task left is to rotate and translate the curve according to (3.64). For reorientation, the path is a single point; therefore, no computation is needed.

After that, an inverse kinematics function is applied to get the joint coordinates and their time derivatives. The next step is to perform checks to see if the **E** point of the robot is in the workspace, if the joint positions $q_1, \ldots, q_5$ are within its limits, and if the kinematic limits for the upper arm axes are not exceeded.

The final step in the generation process is to transform the joint coordinates into suitable machine coordinates because the control system uses different units and zero offset than the kinematic model shown in Fig. 2.3. The main output of this function is a variable `Q_machine` containing the joint positions q_i and velocities $\dot{q}_i$, $i = 1 \ldots 5$, which can be loaded into the physical robot, and the trajectory can be performed. The rest of the output variables serve for plotting the trajectory, which is used especially in the user interface for this generator, described below.

3.6 User interface

By using the functions described above, one can generate the desired trajectory. However, to simplify the generation process, a user interface was created using Matlab App Designer [17]. It uses all the functions described above, with only a few changes to make it compatible with the rest of the user interface.

It consists of three pages. On the first page (see Fig. 3.11), the segments are created. The user can fill in the data about every point, and it gets stored in the table on the left. A live preview of the points in space is shown in the 3D plot on the right side. After the user adds all the points to make the desired trajectory, the function `generate_segments` is called, and the segments are generated. In addition, the path and the orientation of the wrist at the defining points are shown in the plot.

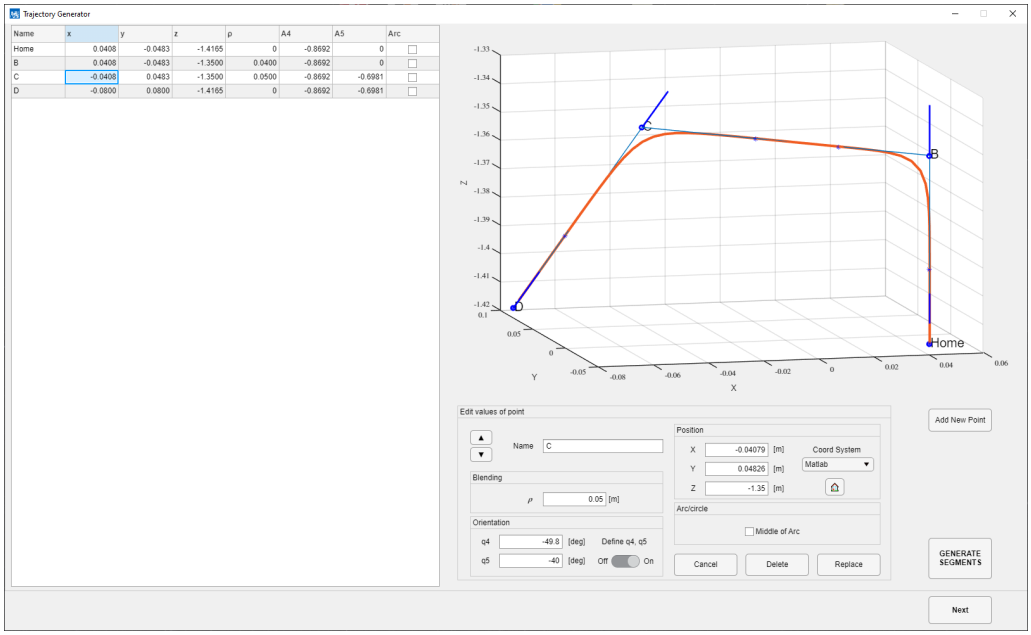


Figure 3.11: First page of the user interface

After that, the user can move to the second page (see Fig. 3.12), where the individual segments are shown in a table. Every segment is automatically named based on the type of the segment and the names of the points defined on the previous page. The kinematic limits for individual segments can be modified as well as the global kinematic limits for axes. In addition, the desired velocity can be defined either at the end of a line segment or in the middle of a blend segment. After setting everything as the user wants, the trajectory is generated using the function `traj_generation`. A velocity profile of the TCP is shown below the table.

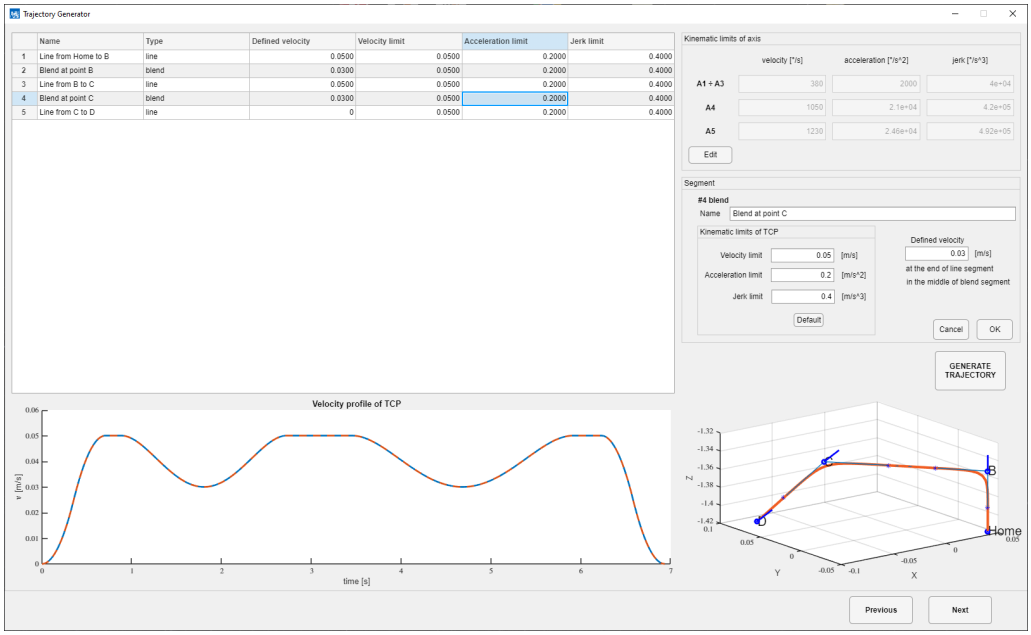


Figure 3.12: Second page of the user interface

A summary of the generated trajectory is shown on the third page (see Fig. 3.13). On the left side, a path colored with respect to the velocity of TCP at that point is plotted. Also, the user can use the buttons below the plot to see the plots of positions, velocities, or accelerations of either the TCP or individual axes. The trajectory can be saved to the computer for later use or be sent directly to the PLC of the robot.

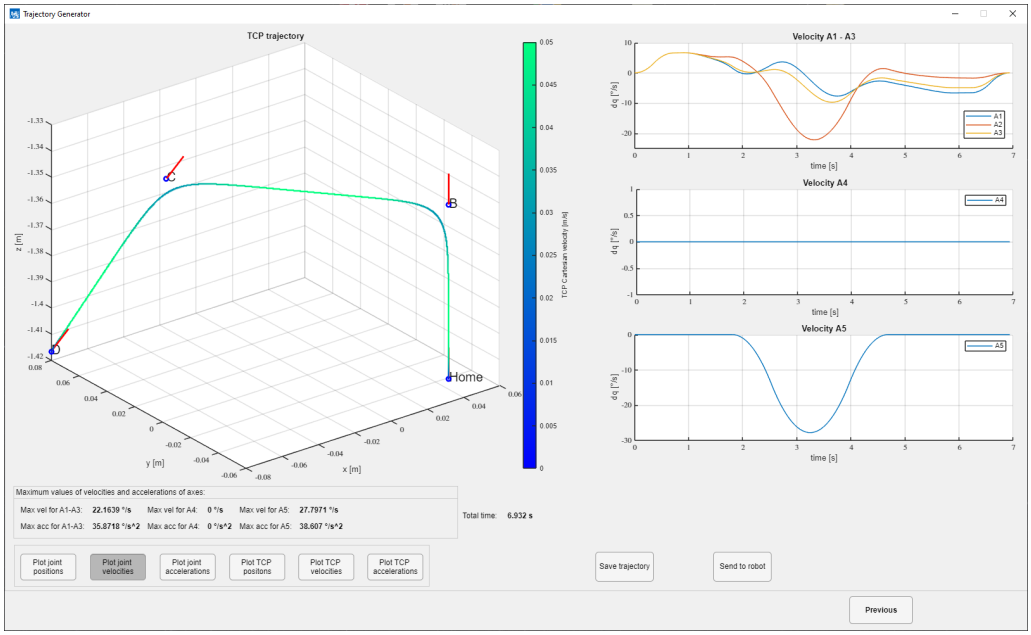


Figure 3.13: Third page of the user interface

Chapter 4

Experiment

To verify that the trajectory planner is working as intended, an experiment with the physical robot was carried out. Three different trajectories were generated using the created user interface of the planner. Then, they were uploaded to the PLC of the robot and performed. Besides visual validation that the robot moves as it should, a Leica laser tracker was used for precise and real-time measurement of the trajectory.

In addition, the positions of the axes were traced by the robot's PLC during the movement, and forward kinematics was used to get the positions of the TCP. Overall, three independent data sets (generated trajectory, laser tracker measurement, and traced positions of the robot's axes) were compared for three different trajectories.

Since the data from the laser tracker are expressed in its own coordinate system, they cannot be compared directly with the data in the coordinate system of the robot. A transformation between the two coordinate systems is quite complicated because the position of the laser tracker with respect to the robot is unknown. Thus, for simplicity, only the relative positions of the measured points are considered, and the comparison is made only on the basis of the shape of the path.

4.1 Setup of the experiment

The following equipment was used to conduct the experiment:

- Leica Absolute Tracker AT960 [18]
- Red Ring Reflector 0.5" [19]
- Reflector Holder 0.5" [20]
- Trajectory Planner (software)
- Leica AT960 Tracker Pilot (software)

The Leica laser tracker was placed in front of the Delta robot, as can be seen in Fig. 4.1. The reflector was then attached to the gripper of the robot using the magnetic reflector holder, as shown in Fig. 4.2. Note that the reflector is not

placed in the TCP of the robot (at the very end of the tool, between the fingers of the gripper). Due to this, the A4 and A5 axes of the robot had to be fixed during the experiment. Then, the shape of the measured path would not be different from the actual path of the TCP.

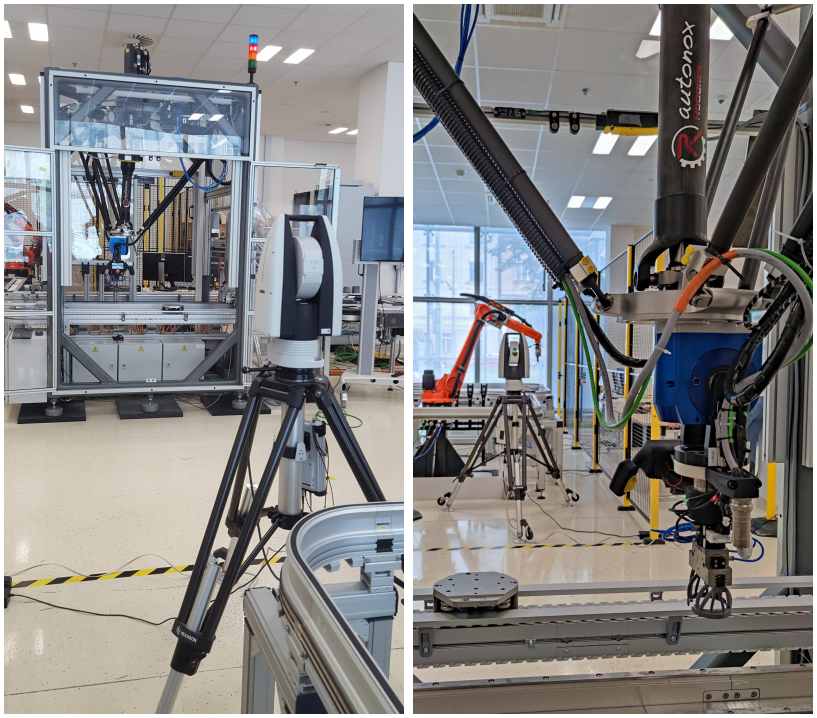


Figure 4.1: Setup of the experiment

The reflector has an acceptance angle of ±30° [19], so the laser tracker was placed as far away from the robot as possible to maximize the size of the measured trajectory.

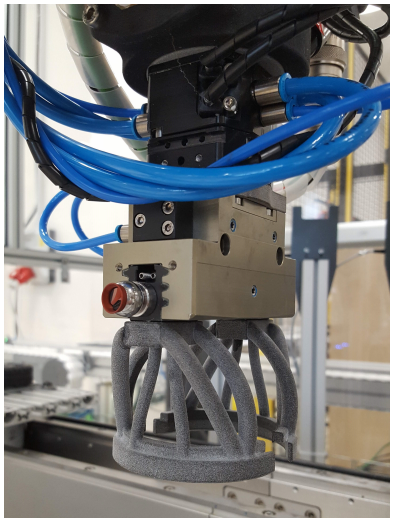


Figure 4.2: Gripper of the robot with attached reflector

The trajectory was generated using the user interface (shown in Section 3.6) and then uploaded to the PLC of the robot via OPC UA (Open Platform Communications – Unified Architecture). The code for sending the trajectory is not part of this thesis and, therefore, is not described in detail. After that, the trajectory was performed, the data from the laser tracker were recorded using a Leica AT960 Tracker Pilot application, and the positions of the axes were traced in the PLC.

4.2 Generated trajectories

Three different trajectories were generated. In all of them, the A4 and A5 axes had to be fixed in their positions due to the reasons with the reflector mentioned above. For each trajectory, tables with all the necessary parameters are given so that anyone can generate the same trajectory according to them. Figures of the geometrical shapes of the trajectories can be found in the following Section 4.3.

The Cartesian kinematic limits of TCP for the three trajectories were set to the following values. Velocity limit $v_{\text{lim}} = 0.05$ m/s, acceleration limit $a_{\text{lim}} = 0.2$ m/s^2, and jerk limit $J_{\text{lim}} = 0.4$ m/s^3. The kinematic limits of the individual axes were left as default.

4.2.1 Pick-and-place trajectory

The first trajectory represents a standard pick-and-place operation and is defined by four points **A**, ..., **D** (see Table 4.1). It consists of three lines with blends between them. The first line is vertical for the pick operation, followed by a horizontal line to move above the final location. The third line is for the place operation, but this time it is not vertical to show a blend between non-right-angled lines (see Fig. 4.4a). In a real application, the gripper could also rotate to match the orientation of the line in space. This could be used, for example, in some assembly applications.

Individual segments with defined velocities are shown in Table 4.2. As mentioned in Section 3.6, the defined velocity is the one at the end of a line segment, whereas, for a blend, it is the velocity in the middle of the blend.

Name	x [m]	y [m]	z [m]	ϱ [m]	q_4 [°]	q_5 [°]
A	0.0408	-0.0483	-1.4600	0	-49.8	0
B	0.0408	-0.0483	-1.3500	0.04	-49.8	0
C	-0.1000	0.1000	-1.3500	0.05	-49.8	0
D	-0.1500	0.1500	-1.4600	0	-49.8	0

Table 4.1: The points creating the pick-and-place trajectory

Type	Defined velocity [m/s]
Line from **A** to **B**	0.05
Blend at point **B**	0.03
Line from **B** to **C**	0.05
Blend at point **C**	0.03
Line from **C** to **D**	0

Table 4.2: Segments of the pick-and-place trajectory

4.2.2 Cuboid trajectories

The other two generated trajectories are defined by eight points **A**, ..., **H**, which are the vertices of a rectangular cuboid. The paths generated by these points can be seen in Fig. 4.5a and Fig. 4.6a. The difference between the two paths is the radius of the blend at the vertices. One path was created with a small blend radius $\varrho = 0.03$ m (see Table 4.3) and the other path with a large blend radius $\varrho = 0.045$ m (see Table 4.4). The difference can also be seen in Fig. 4.3. The individual segments of these two trajectories can be seen in Table 4.5. The defined velocities for each segment are kept the same for both versions.

Name	x [m]	y [m]	z [m]	ϱ [m]	q_4 [°]	q_5 [°]
A	0.008	-0.008	-1.4500	0	-49.8	0
B	-0.008	-0.008	-1.4500	0.03	-49.8	0
C	-0.008	-0.008	-1.3500	0.03	-49.8	0
D	-0.008	0.008	-1.3500	0.03	-49.8	0
E	-0.008	0.008	-1.4500	0.03	-49.8	0
F	0.008	0.008	-1.4500	0.03	-49.8	0
G	0.008	0.008	-1.3500	0.03	-49.8	0
H	0.008	-0.008	-1.3500	0.03	-49.8	0
A	0.008	-0.008	-1.4500	0	-49.8	0

Table 4.3: The points creating the cuboid trajectory with a small blend radius

Name	x [m]	y [m]	z [m]	ϱ [m]	q_4 [°]	q_5 [°]
A	0.008	-0.008	-1.4500	0	-49.8	0
B	-0.008	-0.008	-1.4500	0.045	-49.8	0
C	-0.008	-0.008	-1.3500	0.045	-49.8	0
D	-0.008	0.008	-1.3500	0.045	-49.8	0
E	-0.008	0.008	-1.4500	0.045	-49.8	0
F	0.008	0.008	-1.4500	0.045	-49.8	0
G	0.008	0.008	-1.3500	0.045	-49.8	0
H	0.008	-0.008	-1.3500	0.045	-49.8	0
A	0.008	-0.008	-1.4500	0	-49.8	0

Table 4.4: The points creating the cuboid trajectory with a large blend radius

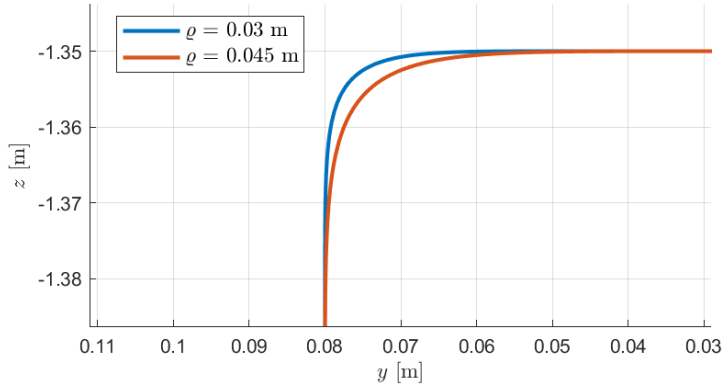


Figure 4.3: Generated trajectories for two different values of the blend radius

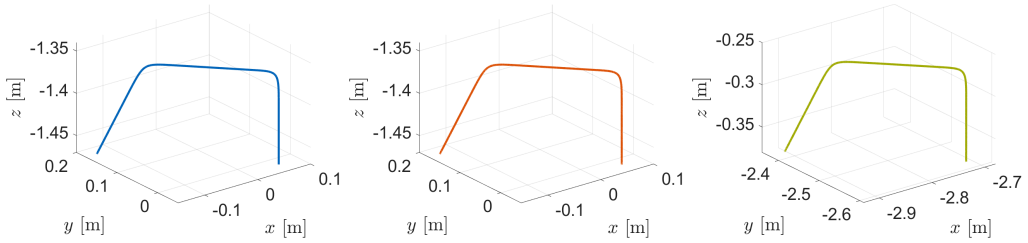
Type	Defined velocity [m/s]
Line from **A** to **B**	0.05
Blend at point **B**	0.03
Line from **B** to **C**	0.05
Blend at point **C**	0.03
Line from **C** to **D**	0.05
Blend at point **D**	0.03
Line from **D** to **E**	0.05
Blend at point **E**	0.03
Line from **E** to **F**	0.05
Blend at point **F**	0.03
Line from **F** to **G**	0.05
Blend at point **G**	0.03
Line from **G** to **H**	0.05
Blend at point **H**	0.03
Line from **H** to **A**	0

Table 4.5: Segments of the cuboid trajectories

4.3 Results

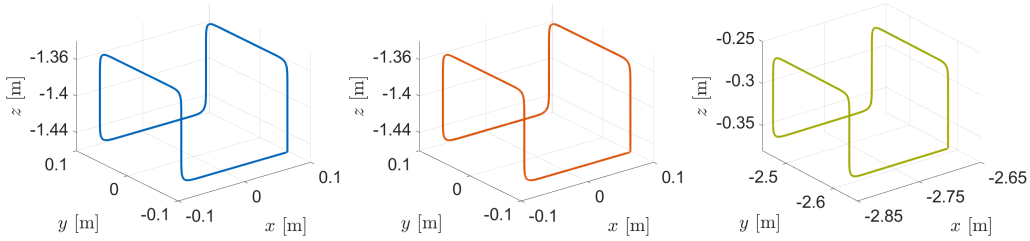
The results of this experiment are presented in the following figures. Data are grouped by the three trajectories, starting with the pick-and-place trajectory in Fig. 4.4, then the cuboid trajectory with a small blend radius in Fig. 4.5 and the last being the cuboid trajectory with a large blend radius in Fig. 4.6. Each figure compares the three different data sources – generated trajectory (blue line), data from the positions of individual axes after using forward kinematics (orange line), and data from the laser tracker (green line).

Note that the measured data from the laser tracker are expressed in a different coordinate system, and the plots are manually rotated to approximately match the orientation of the other two plots of the same trajectory.



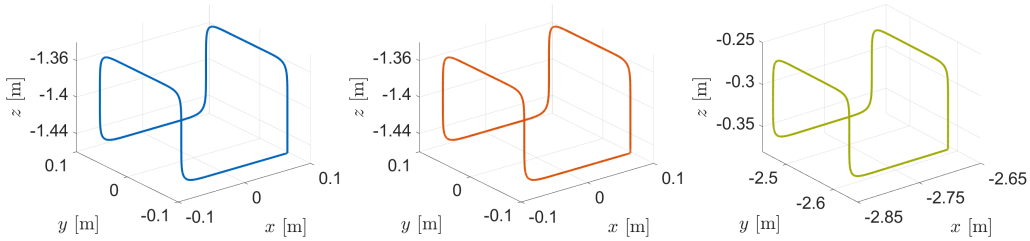
(a) : Generated **(b) :** Forward kinematics **(c) :** Laser tracker

Figure 4.4: Pick-and-place trajectory



(a) : Generated **(b) :** Forward kinematics **(c) :** Laser tracker

Figure 4.5: Cuboid trajectory with a small blend radius



(a) : Generated **(b) :** Forward kinematics **(c) :** Laser tracker

Figure 4.6: Cuboid trajectory with a large blend radius

On the basis of the visual comparison of the plotted trajectories, one can see that the shapes are very similar. Moreover, the generated trajectories (blue lines) and those created by using forward kinematics on the traced axes' positions (orange lines) are identical, which proves that the inverse and forward kinematics match.

The trajectories from the laser tracker (green lines) are not exactly identical, but their shape is similar to that of the other two data sources. Therefore, the trajectory planner is working as intended, and its functionality was verified. Additional experiments and more advanced evaluation methods are needed to obtain more information on the trajectory.

Chapter 5

Conclusion

In this bachelor thesis, a trajectory planner for a 5-axis hybrid parallel-serial Delta robot was created. Thanks to that, trajectories for all five axes of the robot can be performed, which widens the range of possible applications. A custom Pythagorean-Hodograph curve was derived and used for blends between lines. The trajectory planner was created in a way that is easy to use and does not require any programming skills, so anyone can use it. All tasks were successfully completed.

First, a comparison was made between serial, parallel, and hybrid kinematic structures, showing the advantages and disadvantages of each structure. After that, applications of Delta robots were presented based on their key features, such as fast movement and precision.

Furthermore, a detailed description of the hybrid Delta robot located at the CIIRC CTU was given. The mechanical structure of the physical robot and the kinematic model were shown. In addition, the task of forward and inverse kinematics, which had already been solved, was studied, and a brief overview of its derivation was given. The workspace of our Delta robot was also described.

In the following chapter, the trajectory planner for our Delta robot was created. At the beginning of this chapter, used terms, such as path or time scaling, were explained, and the requirements of a trajectory planner were discussed. Next, the used path segments and time scaling functions were described in detail. For the path, a line, arc, blend, and reorientation segments are implemented. An S-curve velocity profile and a quintic polynomial are used for time scaling.

After that, the properties of Pythagorean-Hodograph curves were shown, and a curve to blend between two intersecting line segments with an arbitrary angle between them was derived. Then, this curve was reparametrized, and a quintic polynomial motion law was applied. In addition, formulas for the velocity and acceleration of this blend curve were derived. Also, an issue with velocity and acceleration discrepancies occurred at the merging point of two segments, and a solution was proposed and implemented. At the end of this chapter, the algorithm of the trajectory planner was made by combining the individual functions described above, and a user interface for this planner was created.

An experiment on the physical robot was conducted to verify the functionality of the created trajectory planner. A laser tracker and traced axes positions were used to compare the shape of the real trajectory with the generated one for three

different trajectories. The actual trajectory matches the generated one, which confirms that the trajectory planner works as intended.

This thesis provides a functional trajectory planner for a 5-axis hybrid parallel-serial Delta robot. Future work should focus on extending the possibilities of this planner, for example, by adding new segment types, and improving the existing ones, for instance, by not neglecting the normal acceleration of the TCP. Also, the user interface could be improved by visualizing the point where the kinematic limits are exceeded so that the user knows which segment to modify.

Additional features, such as defining new points of the trajectory in a coordinate system of the attached tool, can be added. The input of the points into the segment generator from a higher-level motion planning system is also an interesting extension of this planner. Finally, the whole planner could be implemented to the PLC of the robot, and the trajectory could be computed in real-time instead of pre-computing it in Matlab.

Bibliography

[1] J. Merlet, *Parallel Robots*. Solid Mechanics and Its Applications, Springer Netherlands, 2012.

[2] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. USA: Cambridge University Press, 1st ed., 2017.

[3] J. Kuřina, "Plánování trajektorie paralelního robota," Bachelor thesis, Faculty of Mechanical Engineering, Czech Technical University, Prague, 2017.

[4] M. Mihelj, T. Bajd, J. Lenarčič, A. Stanovnik, and M. Munih, *Robotics*. Springer Netherlands, 2019.

[5] Midwest Engineered Systems, Inc., "Delta Robots." `https://www.mwes.com/delta-robots/`. Accessed on 2022-07-30.

[6] "Autonox Robotics GmbH. DELTA robot RL5-1450-6kg." `https://autonoxfinder.com/en/a_00802`. Accessed on 2022-07-30.

[7] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics*. Springer London, 2009.

[8] B. Alikoç, V. Šustr, F. Zítek, J. Řehořík, P. Burget, and A. Lomakin, "Motion Modelling and Simulation of a 5-Axis Industrial Delta Robot," *Submitted to Mechatronics*, 2022.

[9] L. Biagiotti and C. Melchiorri, *Trajectory Planning for Automatic Machines and Robots*. Springer Berlin, Heidelberg, 2008.

[10] I. Linkeová, *Základy počítačového modelování křivek a ploch*. 2020.

[11] "3d coordinates of circle center given three point on the circle." `https://math.stackexchange.com/questions/1076177/3d-coordinates-of-circle-center-given-three-point-on-the-circle`, Apr 2016. Accessed on 2022-07-12.

[12] X. Liang and T. Su, "Quintic pythagorean-hodograph curves based trajectory planning for delta robot with a prescribed geometrical constraint," *Applied Sciences*, vol. 9, no. 21, 2019.

[13] T. Su, L. Cheng, Y. Wang, X. Liang, J. Zheng, and H. Zhang, "Time-optimal trajectory planning for delta robot based on quintic pythagorean-hodograph curves," *IEEE Access*, vol. 6, pp. 28530–28539, 2018.

[14] B. Dong and R. T. Farouki, "Phquintic: A library of basic functions for the construction and analysis of planar quintic pythagorean-hodograph curves," *ACM Trans. Math. Softw.*, vol. 41, oct 2015.

[15] R. T. Farouki, "Introduction to Pythagorean-Hodograph curves." `https://faculty.engineering.ucdavis.edu/farouki/wp-content/uploads/sites/51/2021/07/Introduction-to-PH-curves.pdf`. Accessed on 2022-07-15.

[16] MathWorks, "Symbolic Math Toolbox." `https://www.mathworks.com/products/symbolic.html`. Accessed on 2022-07-18.

[17] MathWorks, "Matlab App Designer." `https://www.mathworks.com/products/matlab/app-designer.html`. Accessed on 2022-07-18.

[18] Hexagon, "Leica Absolute Tracker AT960." `https://www.hexagonmi.com/en-US/products/laser-tracker-systems/leica-absolute-tracker-at960`. Accessed on 2022-08-02.

[19] Hexagon, "Red Ring Reflector." `https://shop.hexagonmi.com/na/en_US/USD/Catalog/Laser-Tracker/Reflectors/Red-Ring-Reflector-0-5%22/p/575739`. Accessed on 2022-08-02.

[20] Hexagon, "Reflector Holder." `https://shop.hexagonmi.com/na/en_US/USD/Catalog/Laser-Tracker/Reflector-Holders/Reflector-Holder-0-5%22-%28OFF-10-SH-0-B-12-7-SS%29/p/577220`. Accessed on 2022-08-02.