UNIVERSITY OF TARTU
Institute of Technology
Robotics and Computer Engineering

Carl Hjalmar Love Hult

SurfMotion: An Open Source Pipeline for Robotic Pipe Cutting and Welding

Master's Thesis (30 ECTS)

Supervisor:
Karl Kruusamäe, Associate professor of Robotics Engineering

Tartu 2025

SurfMotion: An Open Source Pipeline for Robotic Pipe Cutting and Welding

Abstract:

Robotic pipe cutting and welding are examples of precise surface-conforming tasks for manipulator robots. Though they are commonplace industrial processes, they pose difficult technical challenges, particularly in motion planning and execution on complex geometries. Existing solutions are largely proprietary, expensive, and closely tied to specific hardware platforms, while open-source alternatives are limited in scope and integration.

This thesis introduces SurfMotion, a ROS package developed to begin to bridge the gap between open-source and proprietary solutions for surface-conforming robotic tasks. SurfMotion provides a modular pipeline for generating, projecting and executing geometry-aware Cartesian trajectories on complex surfaces. The package integrates surface processing, path projection, inverse kinematics feasbility checking, and robot-agnostic trajectory execution, leveraging the ROS 2 and MoveIt frameworks.

The effectiveness of SurfMotion is demonstrated and quantified in terms of Cartesian space error, velocity consistency, and overall smoothness of motion execution through simulated robotic cutting and welding tasks across four example robot models (UR5, UR20, Fanuc m10ia and Kuka iiwa14). Experimental results show that two of the MoveIt motion execution frameworks integrated achieve sub-millimeter accuracy in path tracking.

Keywords: Robotic pipe cutting, Robotic welding, ROS 2, MoveIt, Cartesian motion planning

CERCS: 125 - Automation, robotics, control engineering

SurfMotion: avatud lähtekoodiga lahendus robotiseeritud torulõikuseks ja keevitamiseks

Lühikokkuvõte:

Robotite abil lõikamine ja keevitamine on näited täpsetest, pinnaga kohanduvatest ülesannetest manipulaatoorrobotite jaoks. Kuigi need on tööstuses tavalised protsessid, esinevad nende puhul keerulised tehnilised väljakutsed, eriti liikumiste kavandamisel ja teostamisel keerukatel geomeetriatel. Olemasolevad lahendused on enamasti patenteeritud, kallid ning tihedalt seotud konkreetsete riistvaraplatvormidega, samal ajal kui avatud lähtekoodiga alternatiivid on piiratud ulatuse ja integreeritavusega.

See magistritöö tutvustab SurfMotion'it - ROS-paketti, mis on loodud selleks, et hakata vähendama lõhet avatud lähtekoodiga ja patenteeritud lahenduste vahel pinnaga kohanduvate robotülesannete lahendamisel. SurfMotion pakub modulaarset torujuhet geomeetriateadlike trajektooride genereerimiseks, projitseerimiseks ja täitmiseks keerulistel pindadel. Pakett ühendab endas pinna töötlemise, trajektoori projitseerimise, pöördkinemaatika teostatavuse kontrolli ning robotist sõltumatu trajektoori täitmise, kasutades ROS 2 ja MoveIt raamistikku

SurfMotion'i tõhustust demonstreeritakse ja kvantifitseeritakse Cartesiuse ruumi vea, kiiruse püsivuse ja liikumise sujuvuse alusel läbi simuleeritud torude lõikamise ja keevitamise ülesannete nelja, näidisroboti (UR5, UR20, Fanuc m10ia and Kuka iiwa14) puhul. Eksperimentaalsed tulemused näitavad, et kaks integreeritud MoveIt liikumise täitmise raamistikku saavutavad trajektoori jälgimisel submillimeetrist täpsust.

Võtmesõnad: Robootiline torude lõikamine, robootiline keevitamine, ROS 2, MoveIt, Cartesiuse liikumise kavandamine

CERCS: T125 - Automatiseerimine, robootika, juhtimistehnika

Contents

1. Introduction

High-tech automation in industrial processes has undoubtedly revolutionized modern manufacturing. Tasks like robotic cutting and welding, which demand precise motion of the robot's end-effector along complex surfaces exemplify one of the challenges in advanced manufacturing systems.

Although the industry has a widespread availability of commercial solutions for the planning and execution of robotic paths, these systems are often proprietary, expensive, and closely tied to specific hardware platforms. This is a significant barrier for smaller companies, researchers, and the open-source community.

In response to proprietary solutions, open-source alternative robotics frameworks like ROS and MoveIt have steadily grown in popularity. ROS and MoveIt already tackle some of the industry's challenges, but there remains a significant gap in the availability of comprehensive, geometry-aware pipelines that can handle surface-conforming trajectory planning and execution; capabilities that are essential for tasks like pipe cutting and welding.

The main objective of this thesis is to develop and evaluate an open-source. modular software pipeline that enables surface-conforming Cartesian motion planning and execution, with a particular focus on robotic cutting and welding applications. The system developed aims to address the gap between proprietary and open-source offerings by integrating surface-based path projection, and seamless execution using ROS 2 and MoveIt.

The key contributions of this work are:

- The design and implementation of a robot-agnostic pipeline for geometry-aware trajectory planning and execution.
- Integration of open-source tools for mesh processing, path projection, and feasibility analysis.
- Demonstration and evaluation of the system's performance on representative robotic tasks.

The structure of the thesis is as follows:

- **Chapter 2** provides a review of relevant background.
- **Chapter 3** describes the system architecture and methodology.
- **Chapter 4** presents the experimental results and performance evaluation.
- **Chapter 5** discusses the findings, limitations and potential future work.

AI disclaimer

Large language model tools have been used in the making of this thesis for effectivizing information search (it is often a better search engine than google), translation, conceptually developing ideas and assisting in writing scripts for data processing and plotting.

2. Background

Industrial robotic tasks vary significantly in their motion planning requirements. While pick-and-place operations may very well only require well defined sequential joint-space planning, processes like pipe cutting and welding demand more sophisticated control. These tasks require continuous, precise end-effector motion constrained along curved, convex surfaces, typically achieved with Cartesian-space tracking along with strict orientation control [1].

In the modern industrial robotic environment, robotic cutting and welding tasks are typically addressed with proprietary software solutions provided either directly by the robot manufacturer or a licensed software vendor.
Tools like ABB RobotStudio, Fanuc ROBOGUIDE, Siemens NX CAM, Almacam Weld, allow both programming and simulation of robot operations. They offer high quality integration with specific hardware, but are usually associated with a high licensing cost, and their effective utilization usually demands brand-specific knowledge and expertise.

These tools offer reliability and scalability for large-scale industrial operations, but at the same time, they pose a significant barrier to entry for small-scale enterprises, academic environments and developers seeking open, light and adaptable alternatives for robotic automation. Interest in this alternative is evident in the continued development and maintenance of open-source robot frameworks like ROS, the ROS-industrial suite, and advanced motion planning libraries for robotic arms such as MoveIt.

Achieving the required spatial precision without relying on purpose-built proprietary software is a complex problem, and so the aim of this chapter is to provide the technical context required to understand the pipeline developed in this thesis and explain the main motivations behind its development.

2.1 Glossary

Before anything else, a baseline level of important terms should be established.

End-effector, **tool**, or the abbreviation **EEF** - are all terms used to describe the shape or the device that sits at the end of a robotic arm. Usually this is the part of the robotic arm that we are mostly interested in, and actually interacts with the environment.

Cartesian space - is the three dimensional space where the end-effector's position and orientation are described.

Joint space - is the space defined by all the possible values that the joints of the robot arm can take.

Pose - a 7 degree-of-freedom description of a position and orientation quaternion in Cartesian space.

Path and **Trajectory** - might not immediately seem that different, but here, their meanings are used in accordance with the definitions in *Robotics Modelling, Planning and Control* by Siciliano et al [2].
"A *path* is a pure geometric description of motion" and "a *trajectory* is a path on which a timing law is specified" [2]. The implication is that a *trajectory* is the motion of the end-effector through a *path* through time, constrained by the acceleration of the actuators in its joints.

Motion Planner - is an application that takes a path, plans a trajectory based on the physical constraints of the robot, and maybe even handles the execution aspect.

Tool-path - the term combines the meanings of *tool* and *path*. It specifically refers to the geometric route that the end-effector should follow during a task as defined by a series of way-points.

2.2 Robotic Cutting & Welding

Robotic cutting and welding are conventional industrial processes where precise manipulation of robotic arms is utilized to follow complex three-dimensional paths along object surfaces. In these applications, the robot and its associated software must generate and execute tool-paths that conform to the object geometry (for example the exterior surface of a pipe), with high accuracy. Welding is generally conducted with a tool capable of generating a plasma hot enough to melt and fuse two metallic object together. If the plasma is hot enough, you have a plasma cutter, which is capable of melting metal and cutting it. Robotic cutting can be done either with a plasma cutter, or with a mechanical saw tool. A picture of a welding robot is shown in figure 1.

Figure 1. An industrial welding robot [3]

The main components of a system that tackle this issue should hence contain the following core elements:

- **Surface Representation**: The ability to load and process a 3D mesh or point cloud model of the workpiece.
- **Path Definition and Projection**: A tool to define the cutting or welding pattern, such as a circular seam, and project it accurately on the surface of the workpiece.
- **Pose Generation**: A mechanism for generating feasible end-effector poses along the surface-projected path, considering tool orientation and accessibility.
- **Trajectory Planning**: Integration with motion planners to generate joint trajectories that follow the surface path while satisfying robotic and environmental constraints.

- **Execution**: An interface for sending the planned trajectory to the robot controller, and in some cases, monitoring execution feedback.

2.3 Related works

To the best of the author's knowledge, there is no comprehensive current open-source pipeline that integrates surface-conforming trajectory generation with robot-agnostic execution targeting pipe cutting and/or welding within the ROS 2 scope. Regardless, several open-source projects have been developed to tackle specific sub-tasks. this chapter provides a short introduction to a selection of them.

Cutting

LinuxCNC [4] is a real-time execution system for robotic milling tasks. Featuring a hardware abstraction layer for joint control, it is technically adaptable for most robotic arms. It does not feature high-level functionality like mesh-based trajectory planning.

Welding

The project most resembling the scope of the solution developed in this thesis is the *robotic-welding-demo* [5] project, which presents a welding application for the UR5 robot. The system features a solution for vision-based seam identification, trajectory generation and execution. The execution stack is based on the *urx* library, which is a python library for controlling the Universal Robots family of robots [6]. As a result, the system is not robot-agnostic and does not integrate closely with any ROS-native execution frameworks like MoveIt.

Path generation

Descartes [7] was released as part of the ROS industrial consortium with the explicit aim of addressing robotic welding. Specifically, the task of calculating end-effector poses during welding tasks given an under-defined path. However, the Descartes framework has not been in active development in recent years and has not been ported to ROS 2.

Noether [8] is a tool that provides functionality for mesh-aware path generation. The tool itself runs on ROS 2 Jazzy, as it is largely ROS-independent. It does not feature any built in kinematic feasibility analysis. And as far as the author can tell, focuses on linear toolpaths rather than circular ones.

2.4 Motivation Behind Solution

With this brief preamble of context in consideration, the following specific gaps in whats currently on offer motivate the work behind this thesis:

- Lack of a cohesive, modular solution for geometry-aware Cartesian motion planning and execution available within the open-source tool set.
- Lack of an open-source pipeline for projecting trajectories onto curved surface meshes such as pipes.
- Lack of integrated method to validate the execution feasibility of trajectories with respect to inverse kinematics feasibility, collision avoidance and surface adherence.
- High barrier to entry in proprietary CAM systems, due to high licensing costs and low adaptability.
- High barrier to entry in open-source tool sets, with existing ROS-based solutions requiring extensive manual setup to piece together, limiting usability for non-experts.
- Limited integration between surface-based tool path generation and execution frameworks like MoveIt.

2.5 Tools and Concepts

The remainder of this chapter introduces fundamental concepts and tools used in the implementation of the system. With focus on the building blocks that are leveraged rather than developed from scratch.

2.5.1 Cartesian Motion Planning

Cartesian motion planning is a fundamental concept in robotic manipulator control. It refers to the idea of specifying the desired path of the end-effector in Cartesian space rather than specifying motion in the joint space. It is particularly important in precise tasks like cutting and welding.

Pure joint space planning is useful when the exact end-effector path is not critical for operations, such as when operating in free-space and in tasks like pick-and-place [2]. Joint space motion planning is not necessarily even restricted to point-to-point motions, in fact more complex trajectories can easily be planned by staggering joint states sequentially. However, it is difficult, if not just impractical to achieve the task of carefully planning a tool-path.

In cartesian motion planning we invert the question and formulate the problem as: I want the end-effector to wind up in this specifc pose, what is the path of least resistance to a joint state that will result in this? This is the core problem that is solved with inverse kinematics (IK).

2.5.2 Inverse Kinematics

Inverse kinematics is the process of calculating the joint parameters: θ required so that the end-effector achieves a desired pose $P = \{x, y\}$ in Cartesian space. This explanation is summarized graphically in figure 2.

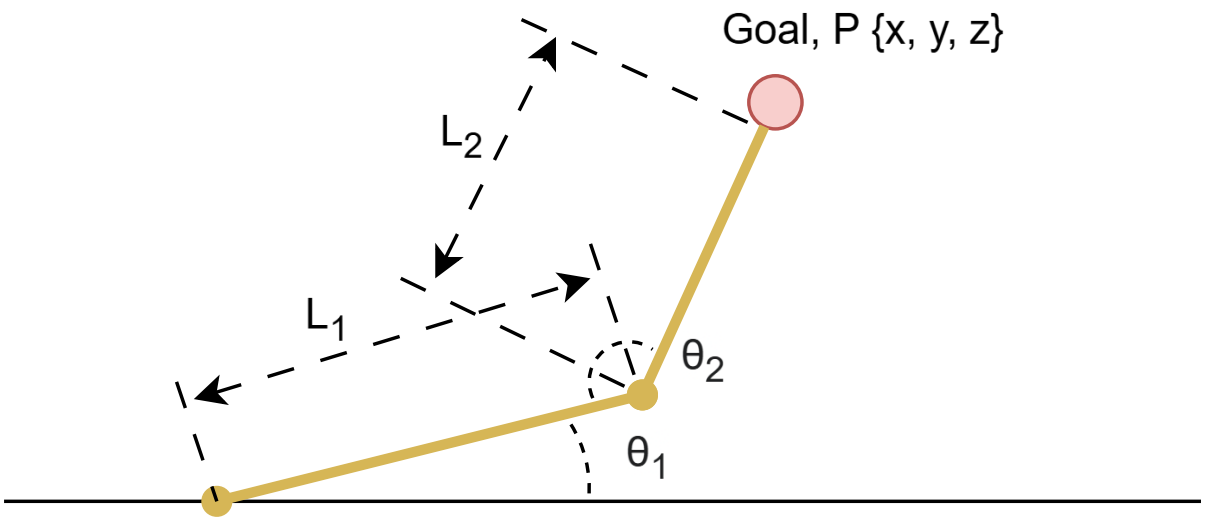


Figure 2. Network with populations and distances

Inverse Kinematics solvers are generally divided into two camps, *analytical* solvers, and *numerical* solvers.

In general, the fewer the degrees of freedom that need to be considered, the more plausible it is to use an analytical solver. An analytical solver would result in a set number of plausible solutions. In the example illustrated in figure 2 is a 2-link manipulator with 2 degrees of freedom. An analytical solver would find one solution to be the joint configuration that is already illustrated, and another where θ_1 is larger and θ_2 would be inverted.

For more complex setups there might exist infinite joint configurations that achieve the same end-effector pose, in which case a numerical solver would be necessary. How exactly the different available numerical solvers arrive at a solution differs, and so do the input parameters.

IKFast [9] is an analytical solver featuring fast and stable solutions. IKFast relies on C++ code that has to be generated specifically for a robot model. While it has not seen recent active recent development, it is integrated in Motion Execution frameworks like MoveIt (see section 2.5.4).

KDL - Kinematics and Dynamics Library) [10] is a widely used numerical solver that works with most robot kinematic chains out of the box. It does not require any code generation like IKFast, but is slower due to being an iterative solver. KDL is the default inverse kinematics solver in MoveIt (see section 2.5.4).

Deterministic vs Non-Deterministic behavior

Inverse kinematic solver behavior could also be classified as deterministic or non-deterministic, depending on how they arrive at a solution for a given end-effector pose.

Analytical solvers like IKFast are deterministic; for any given pose and robot configuration, they will return the same solution. Or in the case of trajectory generation where one might prefer as little joint-movement as possible, they will return the same solution based on some specified criteria, like: *Return the solution that is most similar to the current joint state*.

Numerical solvers like KDL, are more likely to exhibit non-deterministic behavior. KDL specifically will attempt to iteratively solve inverse kinematics by starting from an initial guess (*seed state*) and search for a valid joint state that achieves the desired pose. Due to the possibility of an infinite number solutions, the impact of the current seed state and other factors like floating point number inaccuracies, identical queries for the same pose target may yield different joint solutions. For queries from different seed states it is safe to assume non-deterministic solutions.

In spite of the non-determinism, numerical solvers like KDL are still relevant in Cartesian path planning, and highly relevant for the work in this thesis. KDL is highly adaptable to most robot configurations in contrast to IKFast. Additionally, numerical solvers can be used to explore a range of valid trajectory execution plans. Finally, the non-deterministic aspect can be mitigated for repeatability by saving and storing a trajectory after computation. As long as the seed state can be replicated, the motion plan can likely be executed again.

2.5.3 Surface-Based Path Generation

In cutting and welding tasks on curved surfaces, the end-effector tool must carefully move in adherence to the surface geometry. Cartesian space planning is therefore critical.

A wide variety of methods can be employed to generate a surface-adhering path. The most physically grounded example would be to manually manipulate a robotic arm and recording the joint states to the resolution desired. This is in fact quite common, and research related to technologically improving this process using, for example, augmented reality [11], or just optimizing the path with consideration of the physical limitations of the robot [12] is an actively studied area.

In this thesis, the surface geometry is represented as a point cloud. Each point in the point cloud includes an associated normal vector to the surface. The path is represented as a point cloud in 2 dimension, representing the contour of the cut or the welding *joint?*. In the path-to-surface projection process, the visible part of the point cloud from the perspective of the path is condensed onto a 2D plane parallel to the plane of the path. A 2D nearest-neighbor search conducted for each point in the path, after which the corresponding 3D points on the surface geometry are selected as the projection. This is explained in more depth in chapter 3.1.2.

2.5.4 ROS 2 and the Motion Planning Stack

ROS 2 also referred to as just *ROS* is a widely used framework for standardizing robotics-related software packaging and information flow between packages.
MoveIt 2 [13], also referred to as just *MoveIt* is an extensive motion planning framework for robotic manipulators built on ROS 2. It provides high-level interfaces for inverse kinematics, collision detection, trajectory generation, motion planning and more. The general workflow when using MoveIt for Cartesian path following tasks involves generating a trajectory using a Cartesian path API and then passing it on to the motion execution API.

2.5.5 MoveIt Cartesian Interpolator

The MoveIt Cartesian Interpolator is exposed for applications written for ROS through the *ComputeCartesianPath* interface. It is a component that that interpolates way-points in Cartesian space and computes the corresponding robot joint states using inverse kinematics as a trajectory. Compared to other Cartesian path trajectory planners it is somewhat basic in functionality. It will attempt to compute a trajectory, but if the inverse kinematics solver is ever not able to find a solution to a set number of sequential interpolated points between way-points, it will fail.

2.5.6 Pilz Industrial Motion Planner

The Pilz Industrial Motion Planner is an alternative to *ComputeCartesianPath* as it provides similar functionality, while supporting trajectory smoothing and circular interpolation between

way-points. In theory this should make Pilz Industrial Motion Planner better suited for industrial tasks. Just like with ComputeCartesianPath it is subject to the laws imposed by inverse kinematics, and can fail due to a bad seed-state propagating into the joint trajectory when using non-deterministic solvers.

2.5.7 MoveIt Servo

MoveIt Servo is a control module in MoveIt that enables steering the end-effector through Cartesian space in real-time, either by providing it with velocity or position goal commands. Under the hood, Servo just computes an inverse kinematics solution for the next logical state. It features collision avoidance, and will avoid exceeding joint limits on the fly. This is by no means a guarantee of MoveIt Servo being able to complete trajectories that the earlier motion planners can not, and it is just as much subject to bad seed-state propagation ruining a motion plan.

2.5.8 ROS 2 control controllers

The trajectory execution in this system is handled by the ROS 2 control framework which provides an abstraction layer between planners and actual hardware or simulated hardware. Each planning backend publishes a velocity command which is subscribed to by the corresponding ROS 2 control controller. MoveIt is capable of generating a controller for a given robot using the setup assistant, and for ease of portability, it would be recommended to use this controller. A real driver would have to be used when running a real robot, and the availability of drivers depends on the manufacturer. For now, only the MoveIt controllers are natively supported.

2.5.9 Reach Reachability Analysis

Reach [14], [15] is a ROS 2-compatible tool that allows visualization of the reachability of a specific robot with respect to set of target poses on a workpiece in the form of a mesh. It supports any given robot model given a ROS-standard URDF description and a MoveIt-standard SRDF description, which in the context of this thesis would come straight from the MoveIt setup assistant. For reachability calculations, Reach uses a user-defined inverse kinematic solver packaged by MoveIt, and is capable of exporting a database of the poses and their associated reachability indexes. Reach in action is showed in figure 3.

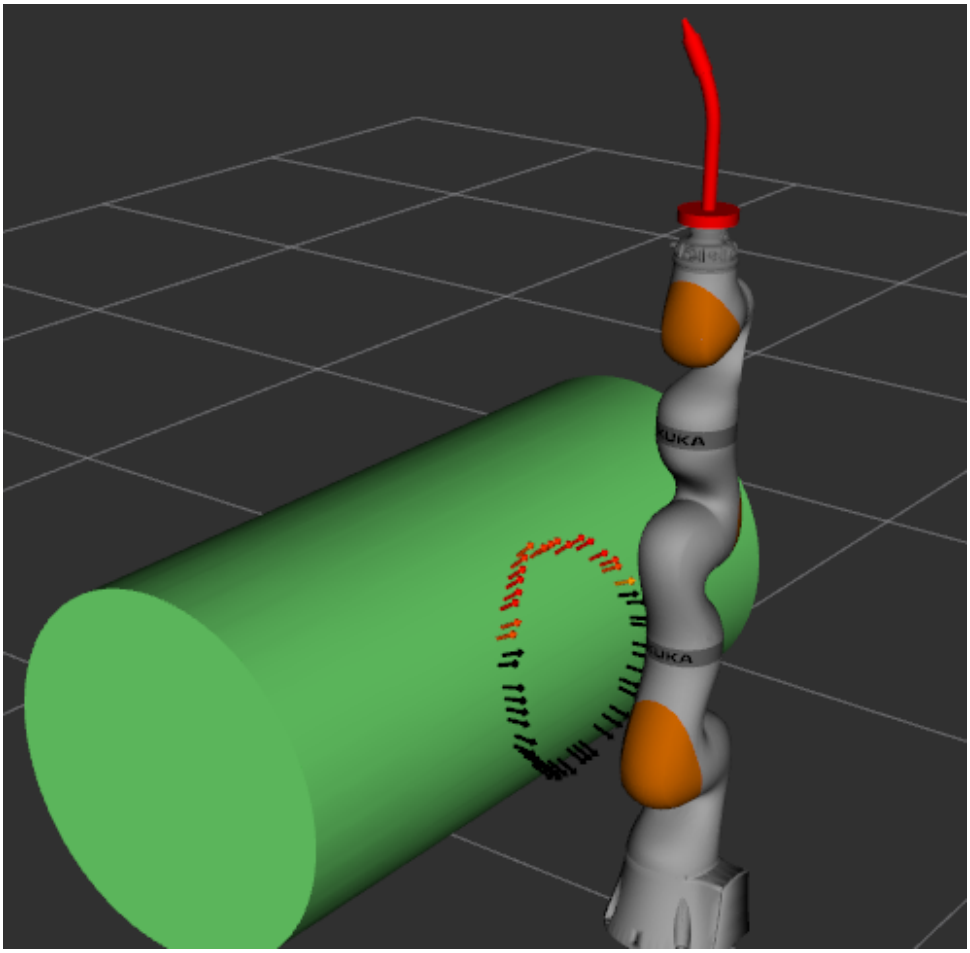


Figure 3. Reach in action. The arrows on the mesh represent way-points, and their color represent the robot's ability to reach them, anything other than black is reachable.

3. System Architecture Overview and Implementation

The solution developed in this thesis is a flexible and robot-agnostic pipeline for generating and executing Cartesian trajectories that conform to a 3D surface. Given a mesh and a reference path, the path can be projected onto the mesh geometry which can be passed on to a motion execution backend to generate a trajectory that can be executed by a robot arm. The solution features a rudimentary graphical user interface for positioning the reference path, and has an integrated reachability analysis provided by Reach ROS for trajectory feasibility evaluation.

For proof of adaptability and evaluation purposes, the system contains three example implementations of different Cartesian movement execution back-ends, including MoveIt ComputeCartesianPath, Pilz Industrial Motion planner and MoveIt Servo. These back-ends can be used interchangeably.

One of the main aims of the solution is to remain independent of any specific robot model. Out of the box, the system utilizes many standard MoveIt abstractions, allowing plug-and-play behavior with different robot models simply by editing a configuration file pointing to the desired robot's MoveIt configuration package.

The following sections describe each component of the system that was built for this thesis in detail, including surface projection, way-point generation, runtime reachability analysis, and execution.

3.1 Software stack

The system is implemented in ROS 2 Jazzy as the core framework, Blender is used for initial mesh modeling, and the Open3D open source 3D processing library [16], handles point cloud processing and other geometric pre-processing tasks. The system utilizes MoveIt 2 for trajectory finalization, inverse kinematics and execution, and finally RViz for visualization. The core components and their relationships are illustrated in figure 4 as a high level overview.

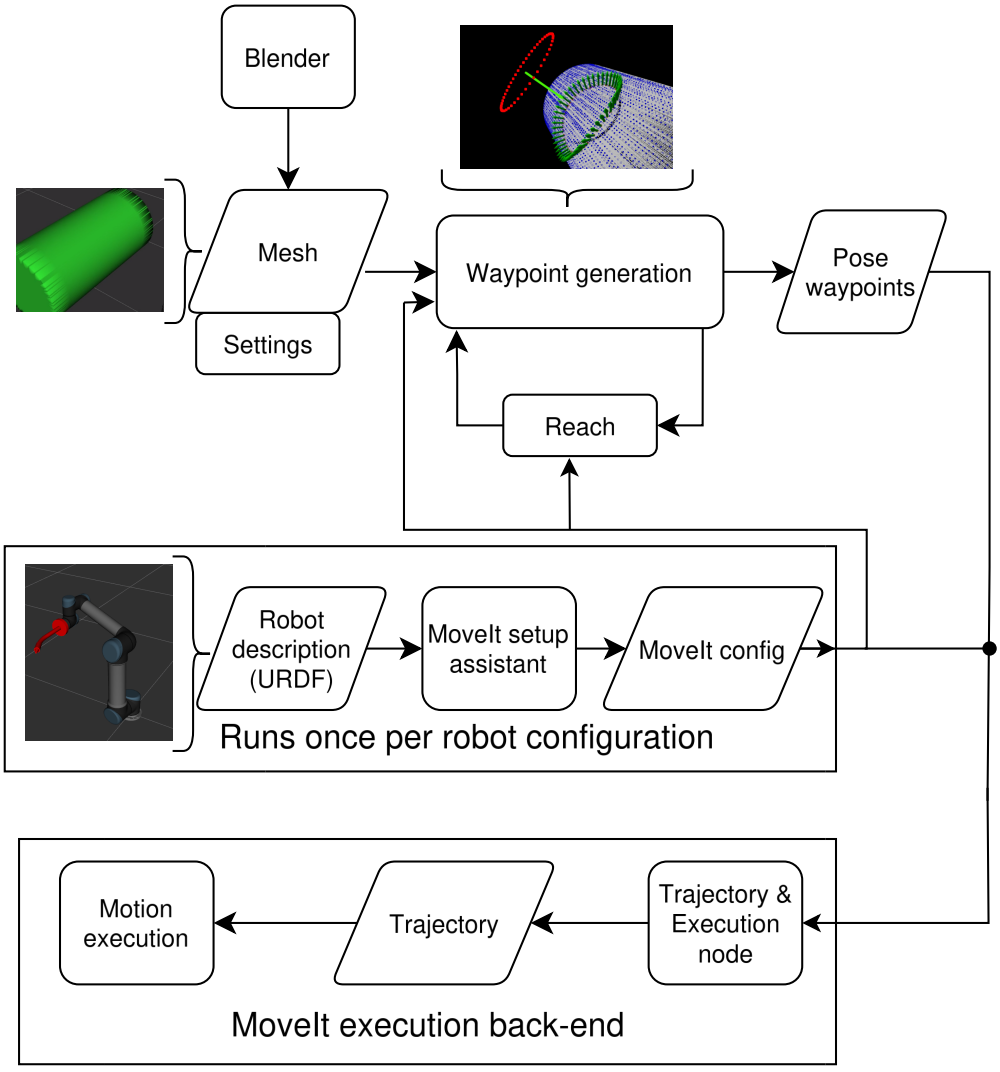


Figure 4. A high level overview of the system architecture and information flows, as well as some visualizations.

3.1.1 Mesh and Point Cloud generation

The system requires a mesh or a point cloud (containing points and normals) describing the geometry of the surface to generate the way-point poses for the motion planner.

In this paper, the open source modeling program Blender was used to make a digital pipe. However, any CAD program capable of exporting a mesh should suffice. For compatibility reasons with ROS Reach, the Polygon File Format *.ply* [17] is used by default for the mesh. Likewise, for compatibility reasons, the point clouds are stored in the Point Cloud Data *.pcd* format [18], specifically in ASCII format rather than binary. The conversion from mesh to point cloud along with normals estimation is handled by the Open3D library.

3.1.2 3D surface projection and Path Extraction

To generate a set of way-points for the path that conforms to the surface geometry of an object, a 2D contour of the cut (this would usually be a circle or an ellipse) is used as a stencil. The aforementioned GUI is used to precisely and visually position the cut contour. It features live updating (on a down-sampled point cloud) of the projection to allow for easy visualization.

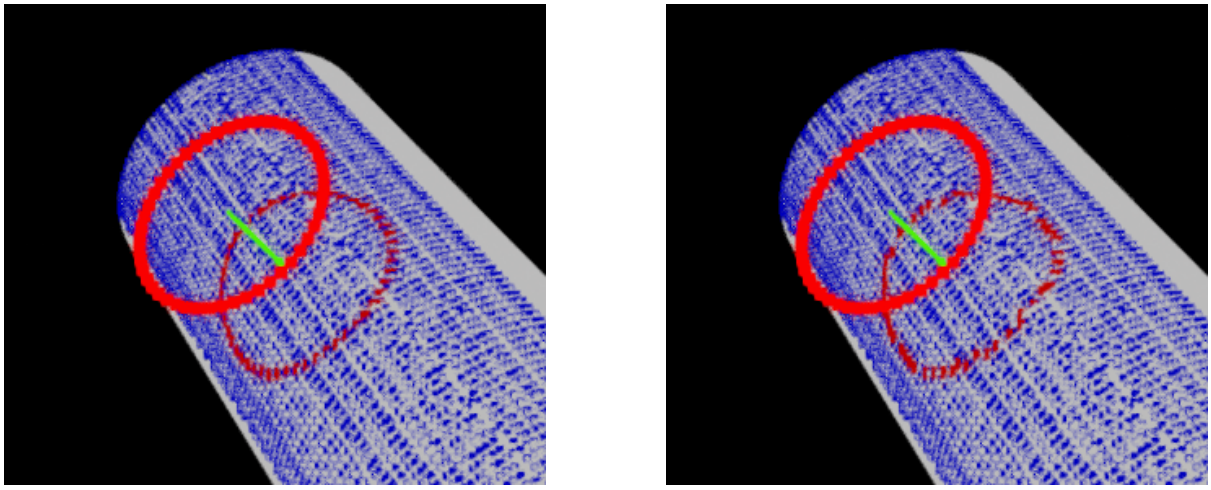
The projection method is initiated by defining a local frame of reference from the perspective of the contour path. The geometry point cloud is then transformed into the local frame of the contour point cloud such that the X and Y axes are aligned with the plane of the contour point cloud. If we let $T_{world}^{contour}$ denote the homogeneous transformation matrix from the contour point cloud frame to the world frame, then each point in the geometry point cloud P_{global} is mapped to that coordinate system as:

$$P_{\text{contour}} = (T_{\text{world}}^{\text{contour}})^{-1} \cdot P_{\text{global}} \tag{1}$$

Once transformed, the 3D points are projected into 2D by discarding the Z-component.

$$P_{\text{contour}} = \begin{bmatrix} x_{\text{ contour}} \\ y_{\text{ contour}} \\ z_{\text{ contour}} \end{bmatrix} \quad \Rightarrow \quad P_{\text{2D}} = \begin{bmatrix} x_{\text{ contour}} \\ y_{\text{ contour}} \end{bmatrix} \tag{2}$$

This results in a distorted 2D projection of the surface geometry that is suitable for nearest-neighbor search. This can be visualized as steps 1 and 2 in figure 6.
Depending on the level of precision required on the projected path to resemble the shape of the contour, more or less points of resolution are required in the contour point cloud itself, by default it is set to 100. Likewise, the geometry point cloud also needs to be more or less dense. By default for a cylindrical pipe with a diameter of 40cm, and a length of 2m, around 2 000 000 points are required to get a satisfactory result, based on empirical testing. The effects of an under sampled point cloud can be seen visually in figure 5.



(a) High density point cloud projection

(b) Low density point cloud projection

Figure 5. Images comparing the effects of the geometry point cloud density.

A naive nearest-neighbor search has a time complexity of $O(n)$ which is not feasible given the number of comparisons needed during projection. To address this, a KD-tree is constructed from the 2D geometry point cloud using the scipy.spatial library, based on the original algorithm by Jon Louis Bentley [19] [20]. After construction, queries average $O(logn)$ time, enabling much faster large-scale nearest-neighbor searches. Each point in the point cloud is matched with a point in the 2D geometry cloud and mapped to a corresponding point in the 3D point cloud, resulting in a projected path.

A configurable offset can also be applied to the projected path to lift it away from the mesh surface. This offset is added along the normal of each point, and is useful for enforcing some level of tool clearance from the actual mesh. For each point P in the projection:

$$P_{\text{offset}} = P + d \cdot n \tag{3}$$

This method allows paths to conform to complex geometries with relatively light computational requirements as long as the region targeted is not heavily self-occluded. However, this is a relatively arbitrarily imposed limitation. By default, points in the geometry that are considered ”not visible” from the perspective of the contour path are filtered out. If one removes this restriction then occluded parts will also be considered. For the purposes of this thesis, simply filtering out ”not visible” points was deemed to result in satisfactory outcomes.

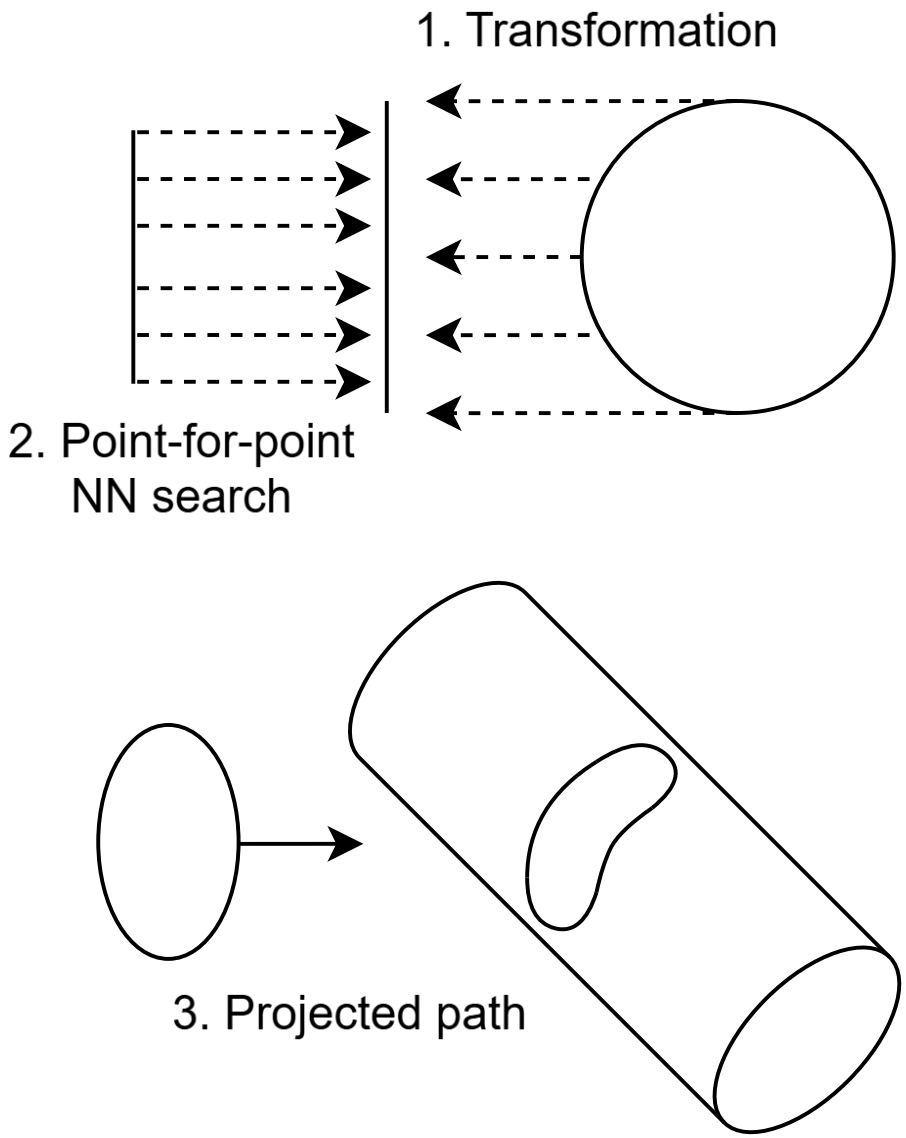


Figure 6. 1 - A curved cylinder face being condensed into two dimensions.

2 - Point-to-point nearest-neighbor search.

3 - Circle to surface conforming shape.

3.1.3 Pose Generation

To generate MoveIt compatible poses, both a position in 3D space and a rotation represented as a quaternion are required. The 3D position can be extracted directly from the geometry. The quaternion can either be derived from the normal vector corresponding to the point. Or it can be extracted from the database that Reach generates after running a reachability analysis. However, the quaternions that Reach generates do not guarantee a consistent tool orientation around its own axis. The information required to be generated for a proper pose can be summarized visually as in figure 7.

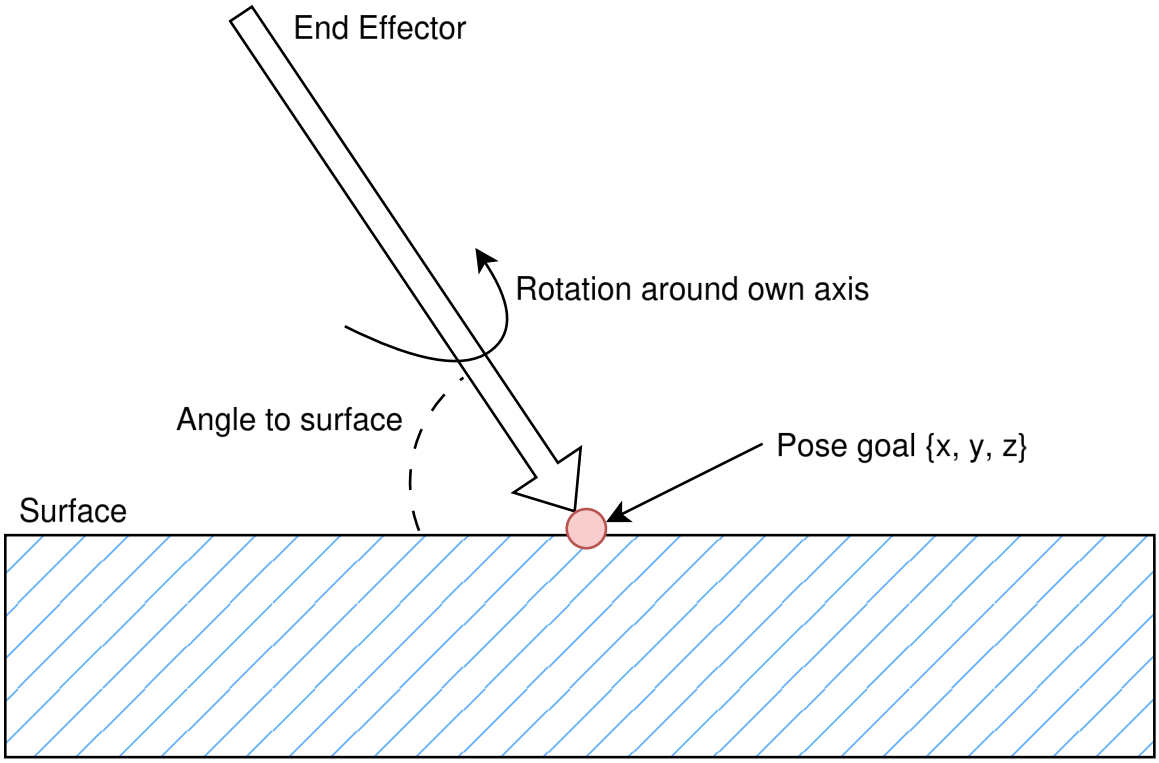


Figure 7. High level concepts related to pose generation that need to be satisfied for proper poses.

In the case of generating a path for constructing a cutting trajectory, the tool is assumed to approach the surface perpendicularly, along the inverse of the surface normal; That is, the normal vector directly defines the direction *from* the surface *toward* the tool. In a welding scenario however, one of the most important parameters is the angle at which the tool approaches the surface [21]. The tool is expected to approach the surface at an angle biased towards the surface of the geometry. To compute the tilt, each surface normal is adjusted by blending it with a vector pointing away from the centroid of the projected path (as illustrated in figure 8). If we let P be

the points in the projection, n be the normals, the outward direction is defined as a unit vector as follows:

$$V_{out} = \frac{P - \text{centroid}}{\|P - \text{centroid}\|} \tag{4}$$

The tilted normals are computed by rotating the normals n by a user-defined angle θ towards V_{out}:

$$n_{tilt} = \cos(\theta) \cdot n + \sin(\theta) \cdot V_{out} \tag{5}$$

This is visually represented in figure 8.

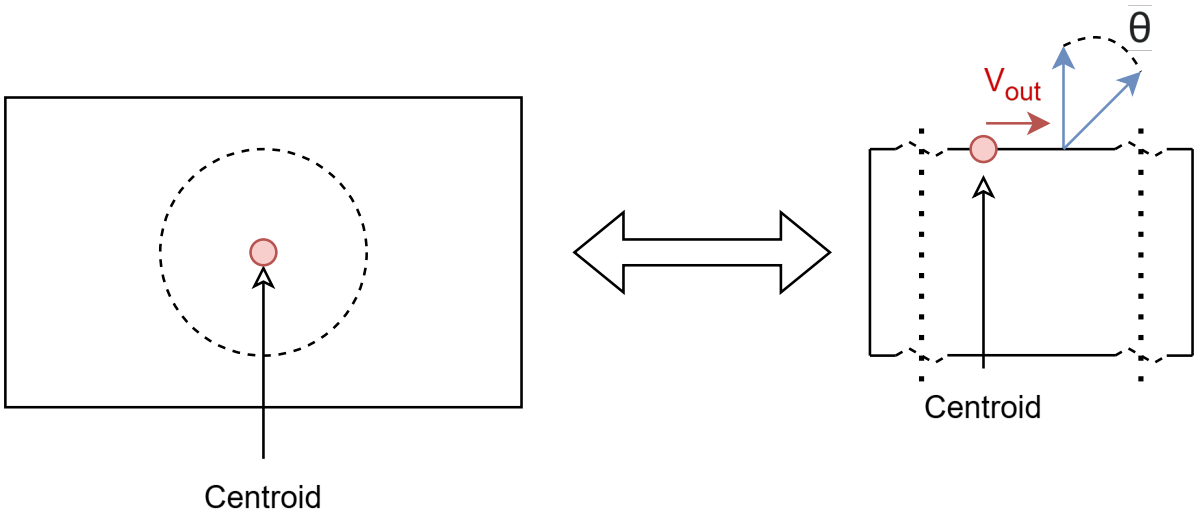


Figure 8. A cylinder with a projected path on it as seen from a top (left) and side (right) view. The normal is tilted out by being blended with a vector V_{out} pointing away from the centroid

3.1.4 Reachability

To allow iterative refinement and feasibility checking of the projected path, Reach is integrated into the system pipeline as a runtime tool for evaluating the kinematic feasibility of the generated poses. Functionally, Reach only expects a point cloud with normals along with a collision mesh and a robot description featuring kinematics. As the projected path is already a point cloud with normals Reach effectively works as a tool to evaluate the path feasibility. If doing a welding analysis, the normals are already forcefully tilted (as explained in section **??**) and it also works effectively for this situation. Based on empirical testing, if Reach classifies all poses in the path

as reachable, the resulting trajectory is highly likely to be successfully planned and executed by MoveIt. While it does not guarantee success since it does not account for trajectory continuity, it effectively filters out paths that are physically impossible for the robot.

3.1.5 Seed State Finder

The problem of non-deterministic inverse kinematics is especially manifested when trying to find a suitable seed state that enables the execution of a full Cartesian trajectory. To the best knowledge of the author, a good seed state would usually have to be found manually. As an additional note on this topic, since it would generally be preferable to have as few joint movements as possible in trajectory execution, this is also something that should be considered.

The approach taken to finding a suitable seed state is the *Seed State Finding* algorithm developed specifically for this software stack. This is partly offered as a ROS-package, but it is also exposed as a library that can be included in any driver node that requires this functionality.

The developed algorithm scans the current MoveIt Scene Graph which contains the all the robot information, as well as any collision objects in the scene. It iteratively conducts brute-force attempts at generating a viable trajectory based on random robot seed states as the end-effector is positioned at the first pose of the path. Under the hood it invokes MoveIt's ComputeCartesianPath which enforces quite strict limits on joint movements. If a seed state is found that allows for generation of a full trajectory the seed state is returned to the main routine which can then set the optimized joint states on the robot.

This algorithm is used in every back-end for generating suitable seed states.

3.1.6 Launch File Structure

ROS provides some useful abstraction tools, and one of them is launch files. Launch files serve to abstract the actual heavy lifting of starting complicated software stacks that might consist of many processes needing to be started at the same time with any number of meta-parameters, and so are they utilized in this solution.

The developed *moveit_ctx_launcher* package provides a main launch file that can be used to launch and set up the context for MoveIt. The user only has to define a meta-information file that points to the MoveIt configuration describing the robot.

The drivers described in the next section (3.1.7) also require some level of context, and specific launch files are also provided in their respective packages.

3.1.7 Planner Integration and Execution Pipeline

After generating a set of surface-conforming, kinematically feasible cartesian poses, they must be converted into a trajectory that can be executed by a robotic arm. This stage is responsible for integrating the pose data into the MoveIt planning framework and selecting an appropriate backend for trajectory computation and execution, MoveIt's ComputeCartesianPath, MoveIt Servo, or Pilz Industrial Motion Planner.

- *MoveIt ComputeCartesianPath* is a straightforward linear interpolator between the way-point poses. It does not feature collision detection beyond self-collisions.
- *MoveIt Servo* enables real-time velocity command streaming and collision detection. Though it requires a control loop backend implemented in the driver node.
- *Pilz Industrial Motion Planner* features both linear and circular interpolation between way-point poses, it also features more advanced constraint handling like joint limits and collision detection. Nominally, it is limited to point-to-point motion plans, but it also features a motion blending API for the purpose of constructing motion plans defined by series of way-points.

Each motion planner is implemented in specific ROS-nodes, hereon referred to as "drivers". Independently of backend selection, the way-point poses are retrieved from a *.json* file generated by the path projection script.

MoveIt ComputeCartesianPath Driver

The system's driver node expects robot meta information which is passed by the user to the driver by invoking it with a ROS launch file. It sets up all prerequisites required for MoveIt ComputeCartesianPath and sets the robot's joints to a suitable seed state using the seed state finder 3.1.5. The only parameters required by the MoveIt ComputeCartesianPath Trajectory planner is a series of way-points and a desired step size. The trajectory planner then returns a feasible trajectory, which is then scheduled for execution by MoveIt's motion planner.

Pilz Industrial Motion Planner Driver

Just like with the ComputeCartesianPath driver, the user invokes the driver by using its associated launch file, pointing it to the robot meta information. The driver configures the robot to a suitable seed state in the same manner as with the last driver. The driver then builds a *MotionSequenceRequest* which is still just a path, but with Pilz-specific meta-information. The

driver then requests the MoveIt sequence action server to calculate and execute a motion plan based on the MotionSequenceRequest, and the motion plan is scheduled.

MoveIt Servo Driver

The MoveIt Servo driver is launched and set up in the same manner as before. The driver then enters the control loop, which constantly queries the current robot state and calculates a velocity request based on the distance to the next way-point in the path. MoveIt Servo is essentially an abstraction layer for streaming velocity commands, so the trajectory has to be calculated on-the-fly by the driver.

3.1.8 Constraint Setup

For setting up the final bit of context in the form of environmental constraints the *object_spawner* package provides an interface for adding collision objects to a MoveIt planning scene. A launch file is pointed to a mesh, and it will be added to the current MoveIt planning scene.

4. Evaluation and Results

The main goal of this evaluation is to demonstrate the adaptability of the developed system across multiple robots and motion execution strategies. The system is tested on different robot models using three distinct execution backends: ComputeCartesianPath, Pilz Industrial Motion Planner and MoveIt Servo. Additionally, this section provides a benchmark of example implementations of MoveIt motion planners for Cartesian space path planning, something that is not well documented.

For quantifying trajectory quality and planner behavior, metrics for as joint-space and Cartesian space velocity, acceleration, and jerk are computed from execution data. There metric provide insight into smoothness and reflect the performance of the tested execution back-ends. The results are presented both numerically and visually using a unified logging and analysis setup appled across robot configurations.

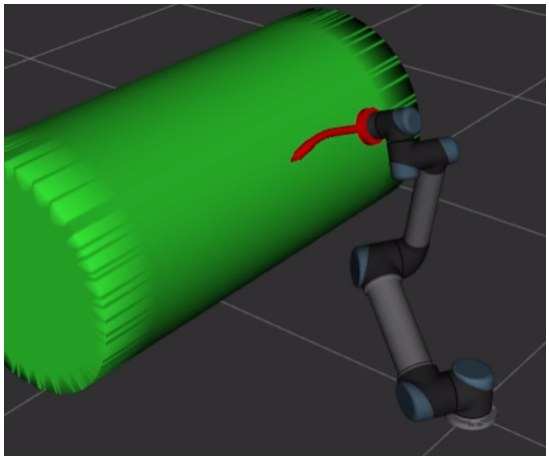
To account for variability in execution speed, both the trajectory duration and the resulting mean end-effector velocity were measured for every run. This information is then used to contextualize and standardize metrics like acceleration and jerk.

4.1 Experimental Setup

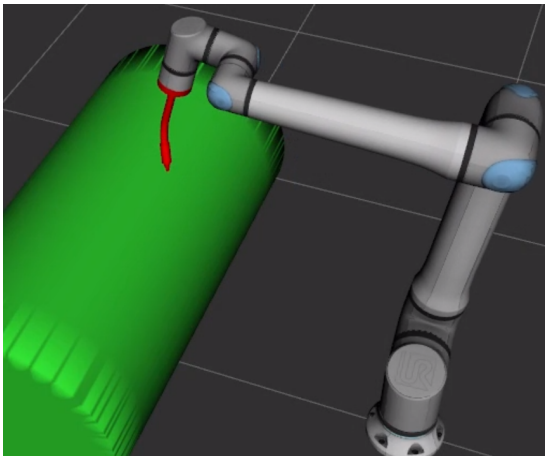
All experiments were conducted within the system's current targeted application, which is a ROS 2 Jazzy and MoveIt 2-based simulation environment. Four robot models were tested, the Universal Robots UR5, Universal Robots UR20, Fanuc m10ia, and Kuka iiwa14. Each robot was defined using a sourced URDF/XACRO that was edited to add a nozzle resembling a welding nozzle to the tool link. From each robot description, a MoveIt configuration was constructed using the MoveIt Setup Assistant. The exact process for one of the robots is documented at: https://github.com/carlhjal/Surfmotion.git.

The only significant change made across all the robots' generated MoveIt configurations' that could impact the final results was adding the previously mentioned artificial velocity and acceleration constraints, as well as a servo_config.yaml that is required for the MoveIt Servo node.

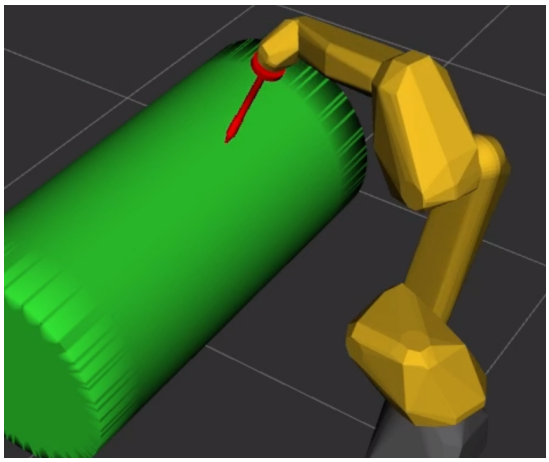
An additional important component of the robot description is a link between the robot base and a common origin for the MoveIt planning interface. In the case of this work, that common origin was arbitrarily decided to be called *world*. If a link was not already defined in the URDF



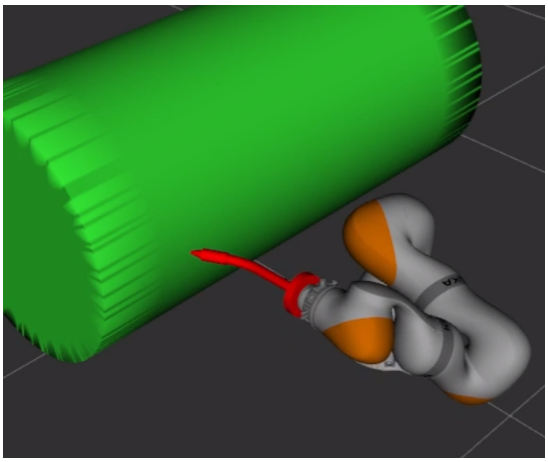
(a) Universal Robots UR5 - UR5



(b) Universal Robots UR20 - UR20



(c) Fanuc m10ia - Fanuc



(d) Kuka iiwa14 - Kuka

Figure 9. Images of the robots simulated in the evaluation along with the abbreviations used to refer to them.

description, a virtual link was added either with the MoveIt Setup Assistant, or manually as an entry in the generated MoveIt config package SRDF file.

While strict velocity control was not enforced, the combination of standardized joint limits, per-robot evaluation, and actual execution time still allows for comparison of planner behavior. The ultimate goal of this evaluation is not to determine absolute performance, but to quantify how different back-ends behave under similar constraints and on a robot-by-robot basis.

4.2 Additional Evaluation Notes

The robots tested express wide differences in kinematic constraints. For example, the UR5 robot is heavily limited in possible tasks due to its small size compared to the UR20 and Fanuc

m10ia, and the Kuka iiwa14 has comparatively strict joint limits compared to the other robotic manipulators. In effect, this means that there is no path that will be entirely executable for all robots, or if there is one, it is hard to identify. Fanuc m10ia and UR20 have enough similar kinematics constraints were tested on the same path.

Joint state data was collected from the /joint_states topic which publishes a message data at a rate of about 100 Hz with some slight deviation. This data is also timestamped which accounts for deviations.

Data about the end effector end-link was collected from the /tf topic. By default the rate of the messages on this topic are somewhat undefined based on empirical data. A workaround for this was to force the robot state publisher to publish messages at higher rates in multiples of two. For the UR robots 400 Hz resulted in tf data being published at 100 Hz, for Kuka iiwa14 200 Hz sufficed. Also the tf data is timestamped.

Since the initial joint state of the robot influences the resulting motion plan, multiple runs were recorded for each scenario with randomized starting configurations. The Cartesian end-effector end-link poses, while they might differ slightly between runs, were very similar between runs. For this section, the end effector poses are averaged over multiple trial runs. Unlike the end-effector trajectories, the specific joint-state data may differ significantly between runs due to variations in initial seed states. As such, they can not be directly averaged between runs. Instead, all run-specific joint-related data is handled as Root-mean-square (RMS) scalar values which are then averaged across runs.

4.2.1 Data Processing

The data collected is in the form of time time-series data of joint positions in radians, and end-effector Cartesian poses during execution. While velocity, acceleration and jerk could in principle be extracted using Euler approximations, this proved too sensitive to noise. Instead, the implementation of Savitzky-Golay filtering in the Scipy [20] library was employed. This method instead fits a polynomial to a window of data, and computes derivatives analytically. As the sampling rate of the data was 100 Hz, a window size of 25 was chosen, representing a time-span of 0.25 seconds. A comparison between simple Euler approximation for derivatives and Savitzky-Golay filtering is presented in figure 10.

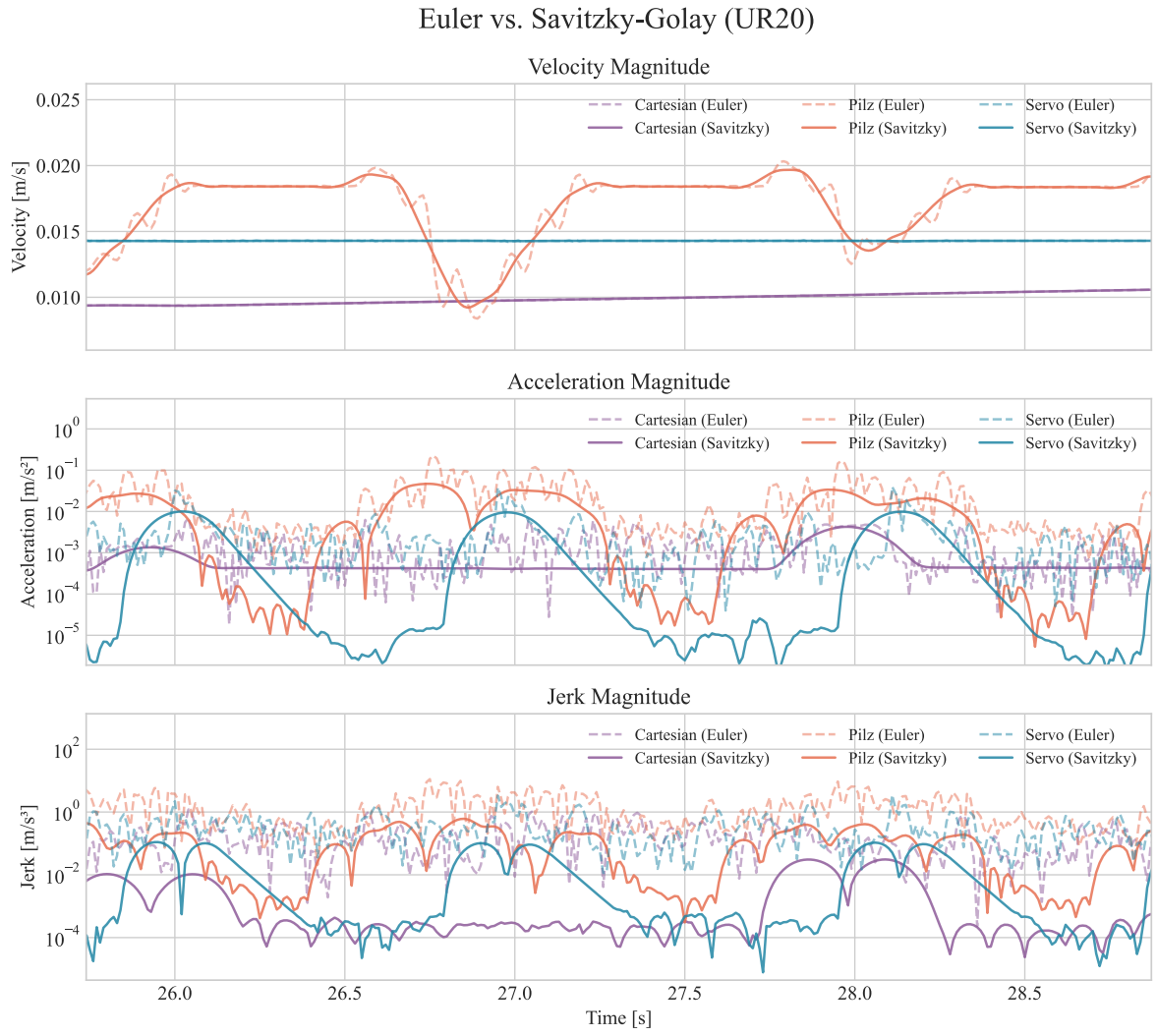


Figure 10. Comparison between Euler approximation and Savitzky-Golay method for extracting kinematic derivatives.

All motion derivatives are calculated this way in the results section.

4.3 Evaluation Setting

The following metrics were tested and inferred from the data collected:

- Cartesian error between planned and executed end-effector positions.
- Smoothness, as a function of velocity-adjusted jerk.
- Consistency of Velocity in Cartesian translational space.
- Overall execution back-end performance score.

4.4 Results

This section is dedicated to the processed results gathered.

4.4.1 Cartesian Error

For calculating the Cartesian error of motion planners, an RMS error metric was calculated for each robot and each run. This error metric was then averaged across all runs.

The results are presented in the form of translation error: 11 and angular error: 12. The data is also presented numerically in table 1.

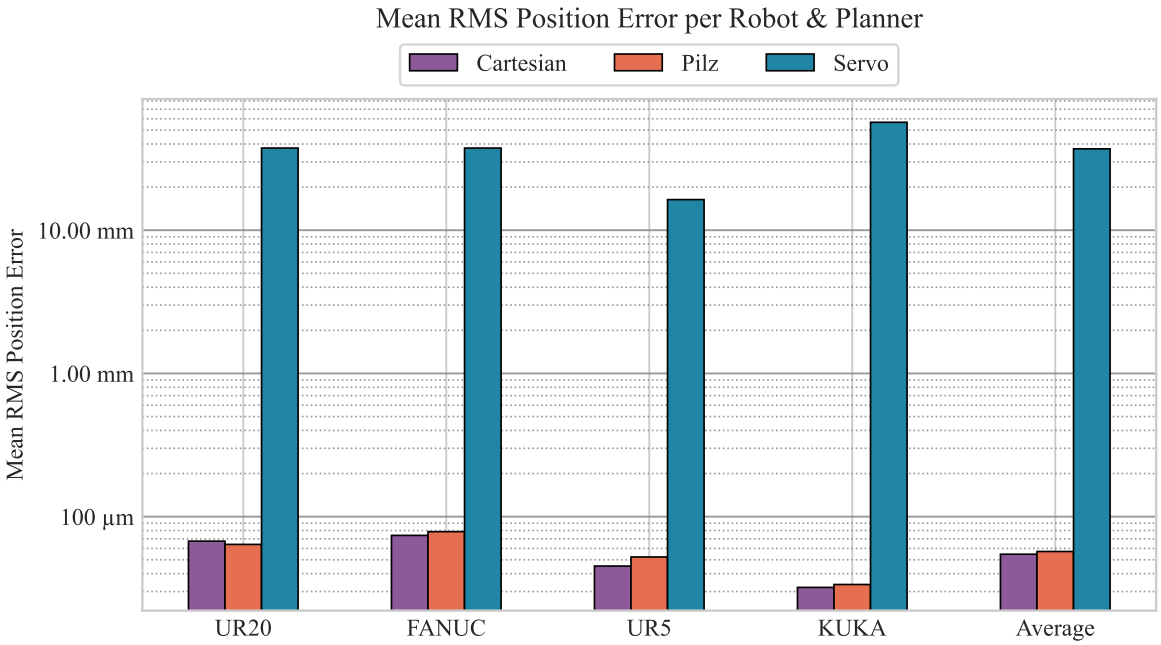


Figure 11. Translational error between the planned and executed path per robot and motion planner. Numeric data in table 1

The recorded average trajectories are presented as a 3D plot in figure 13.

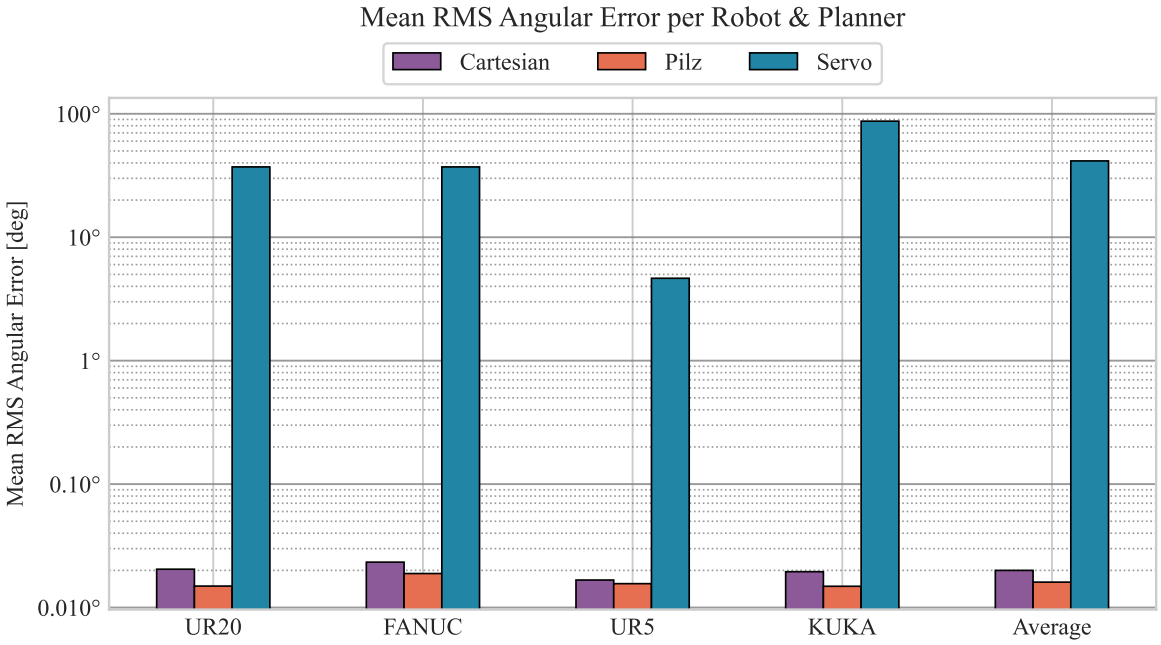


Figure 12. Angular error between the planned and executed path per robot and motion planner. Numeric data in table 1

Table 1. Mean RMS position and angular error for each robot and motion planner

Robot	Cartesian [m]	Pilz [m]	Servo [m]	Cartesian [deg]	Pilz [deg]	Servo [deg]
UR20	6.75e-05	6.4e-05	0.0375	0.02	0.01	37.13
FANUC	7.4e-05	7.86e-05	0.0375	0.02	0.02	37.13
UR5	4.52e-05	5.23e-05	0.0164	0.02	0.02	4.65
KUKA	3.21e-05	3.36e-05	0.0567	0.02	0.01	87.15
Average	5.47e-05	5.71e-05	0.0370	0.02	0.015	41.515

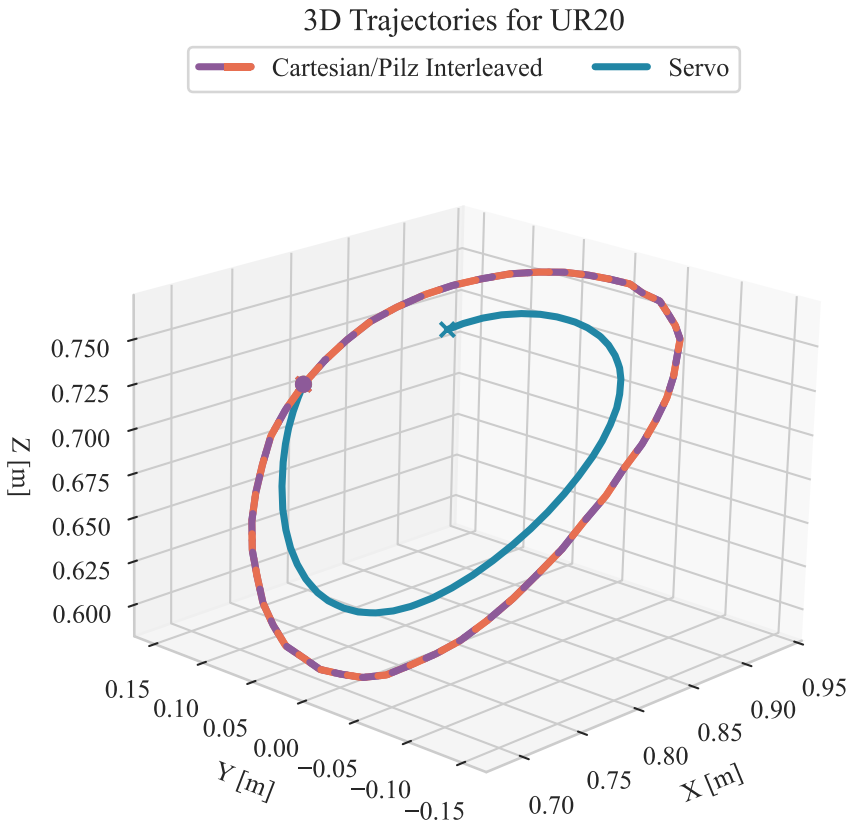


Figure 13. Average 3D trajectories executed by the three different motion planners on the UR20

4.4.2 MoveIt Cartesian Path Planner Motion Profile

The standard step size was set to 5 mm for cartesian path linear interpolation.

The mean velocity, acceleration and jerk RMS values in the joint space are presented in figure 14, giving an overview image of joint activation. Cartesian space motion profiling is divided in translational and angular constituents in figure 15.

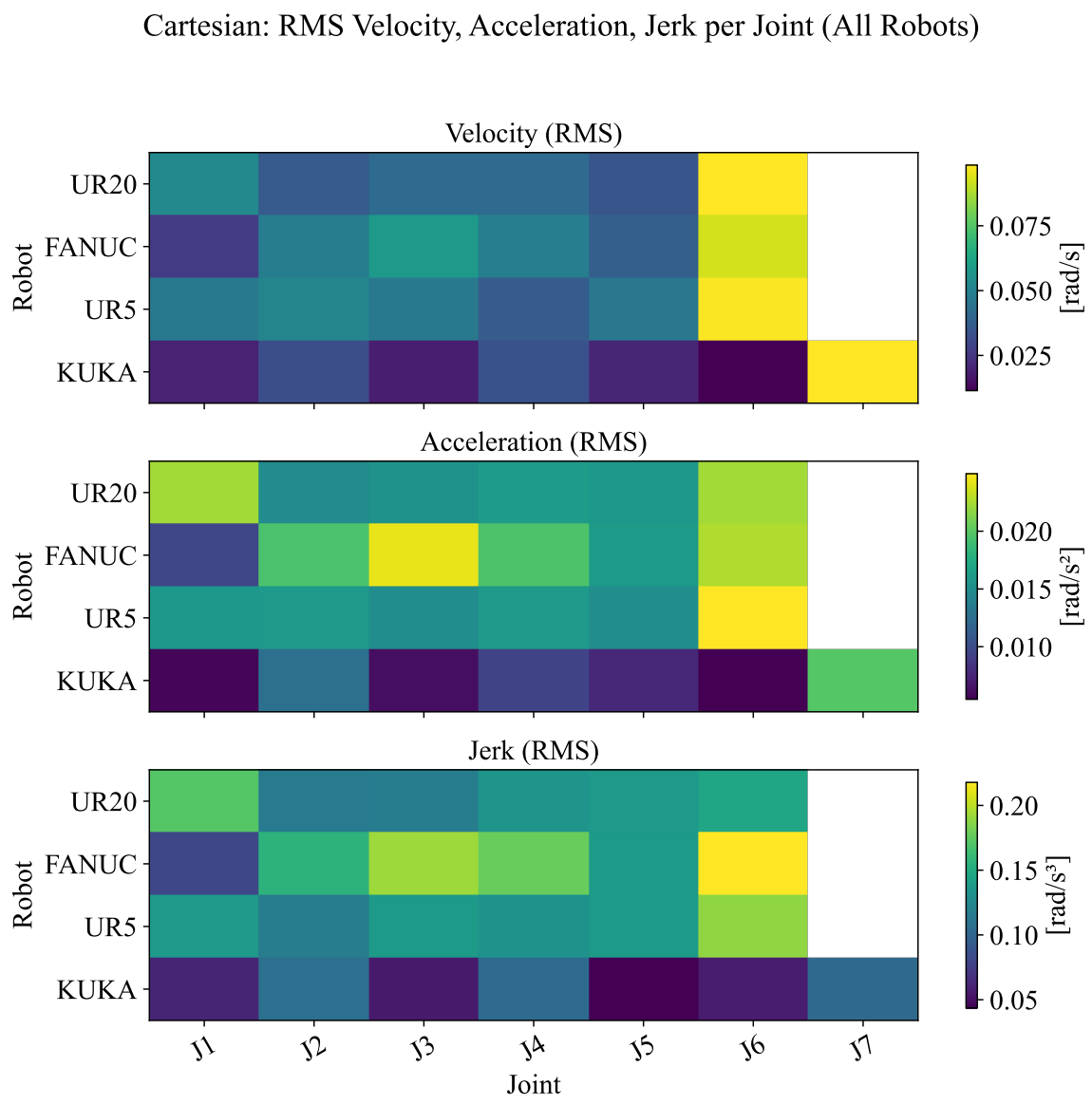


Figure 14. Mean RMS values of Velocity, Acceleration and Jerk on each joint on each robot with the ComputeCartesianPath planner. The y-axes shows different robots, the x-axes indicates joint indices from base to end-effector, and color intensity represent the magnitude of the calculated RMS value

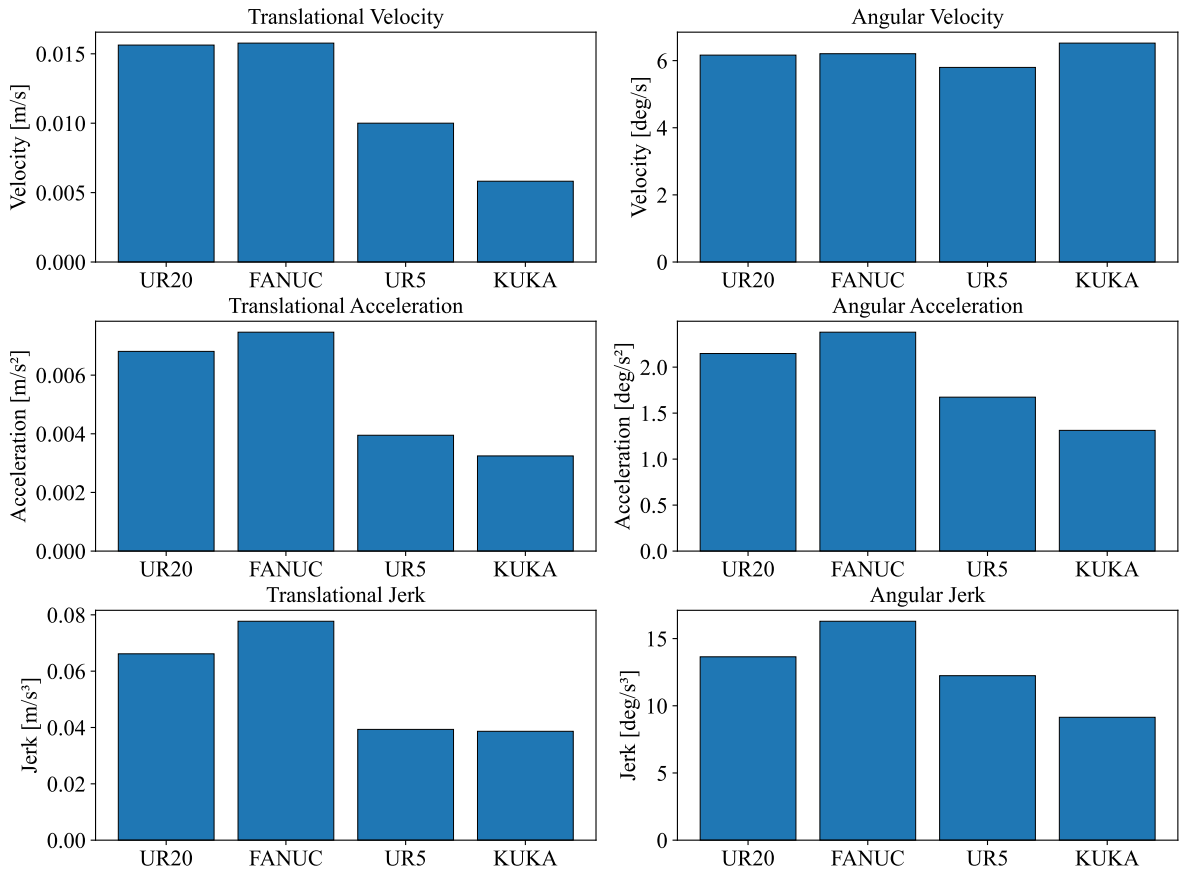


Figure 15. Cartesian space mean RMS metrics for the ComputeCartesianPath planner. Cartesian space metrics are divided in translational and angular constituents.

4.4.3 Pilz Industrial Motion Planner Motion Profile

The mean velocity, acceleration and jerk RMS values in the joint space are presented in figure 16. Cartesian space motion profiling is divided in translational and angular constituents in figure 17.

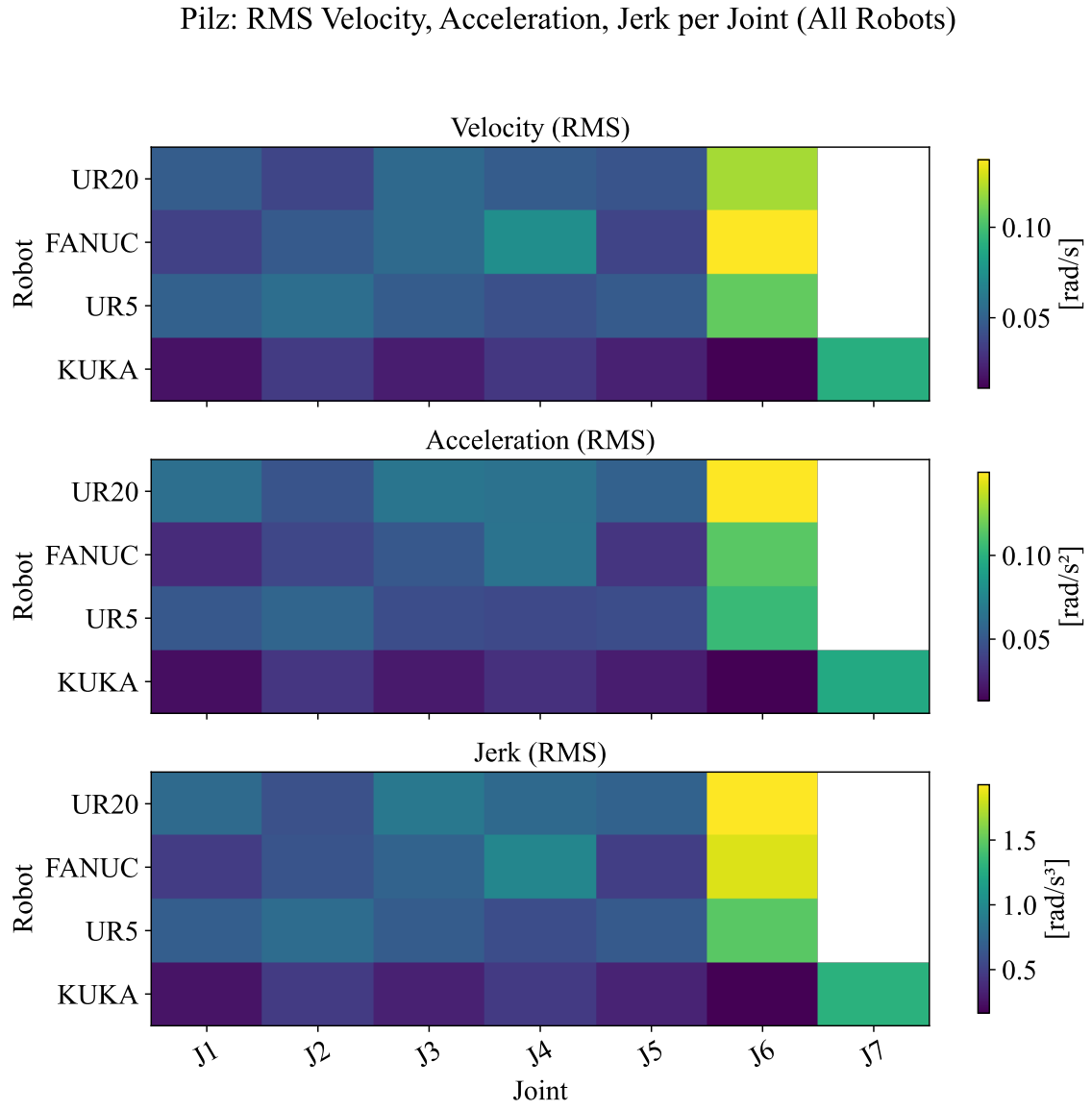


Figure 16. Mean RMS values of Velocity, Acceleration and Jerk on each joint on each robot with the Pilz Industrial Motion planner. The y-axes shows different robots, the x-axes indicates joint indices from base to end-effector, and color intensity represent the magnitude of the calculated RMS value.

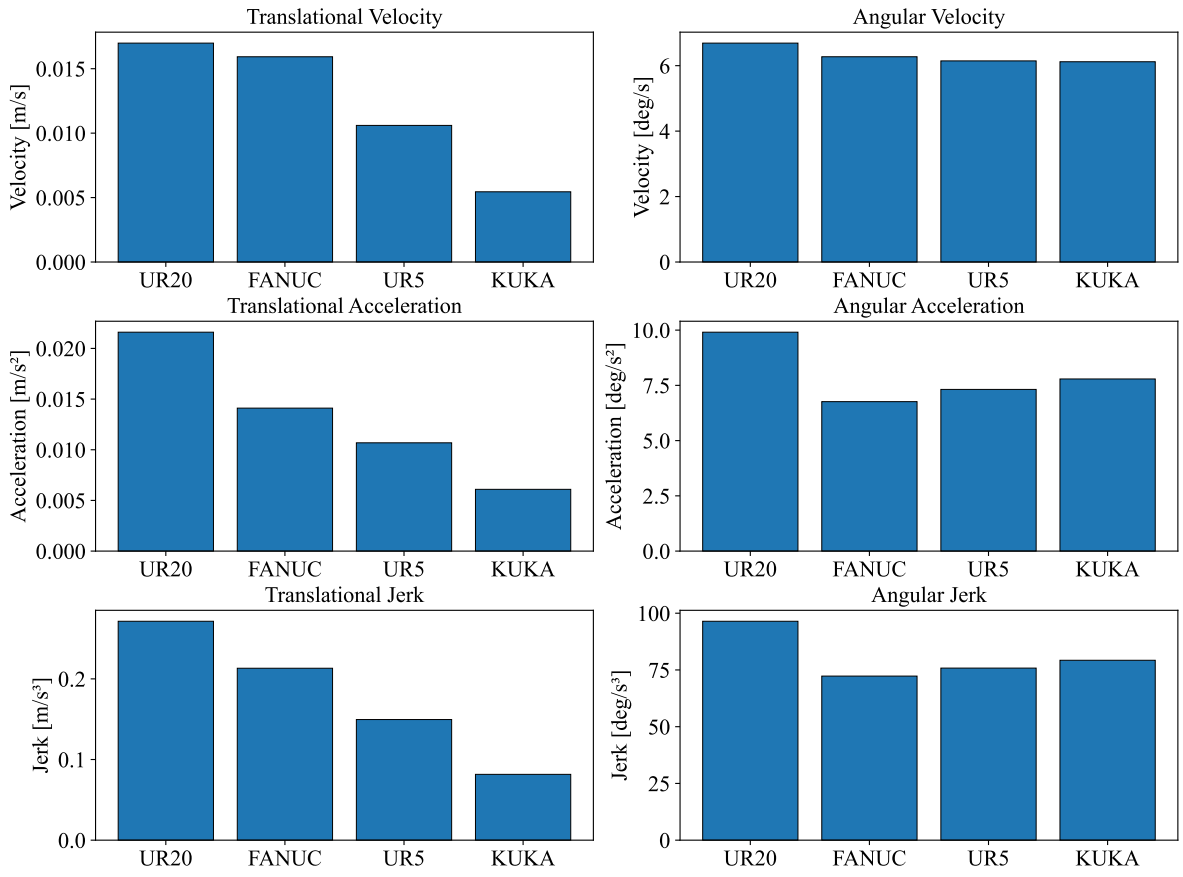


Figure 17. Cartesian space mean RMS metrics for the Pilz Industrial Motion planner. Cartesian space metrics are divided in translational and angular constituents.

4.4.4 MoveIt Servo Motion Profile

The mean velocity, acceleration and jerk RMS values in the joint space are presented in figure 18. Cartesian space motion profiling is divided in translational and angular constituents in figure 17.

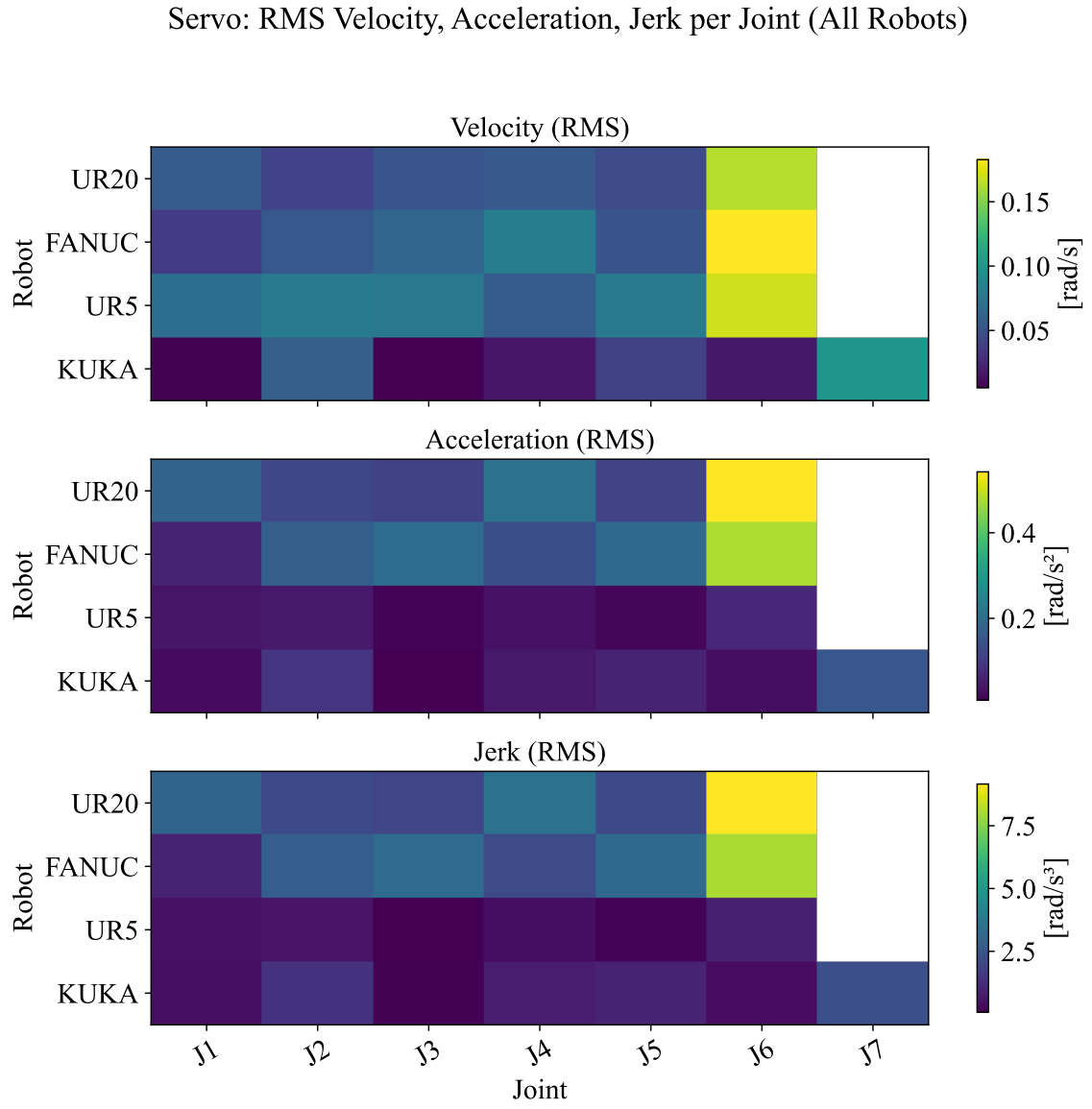


Figure 18. Mean RMS values of Velocity, Acceleration and Jerk on each joint on each robot with MoveIt Servo. The y-axes shows different robots, the x-axes indicates joint indices from base to end-effector, and color intensity represent the magnitude of the calculated RMS value.

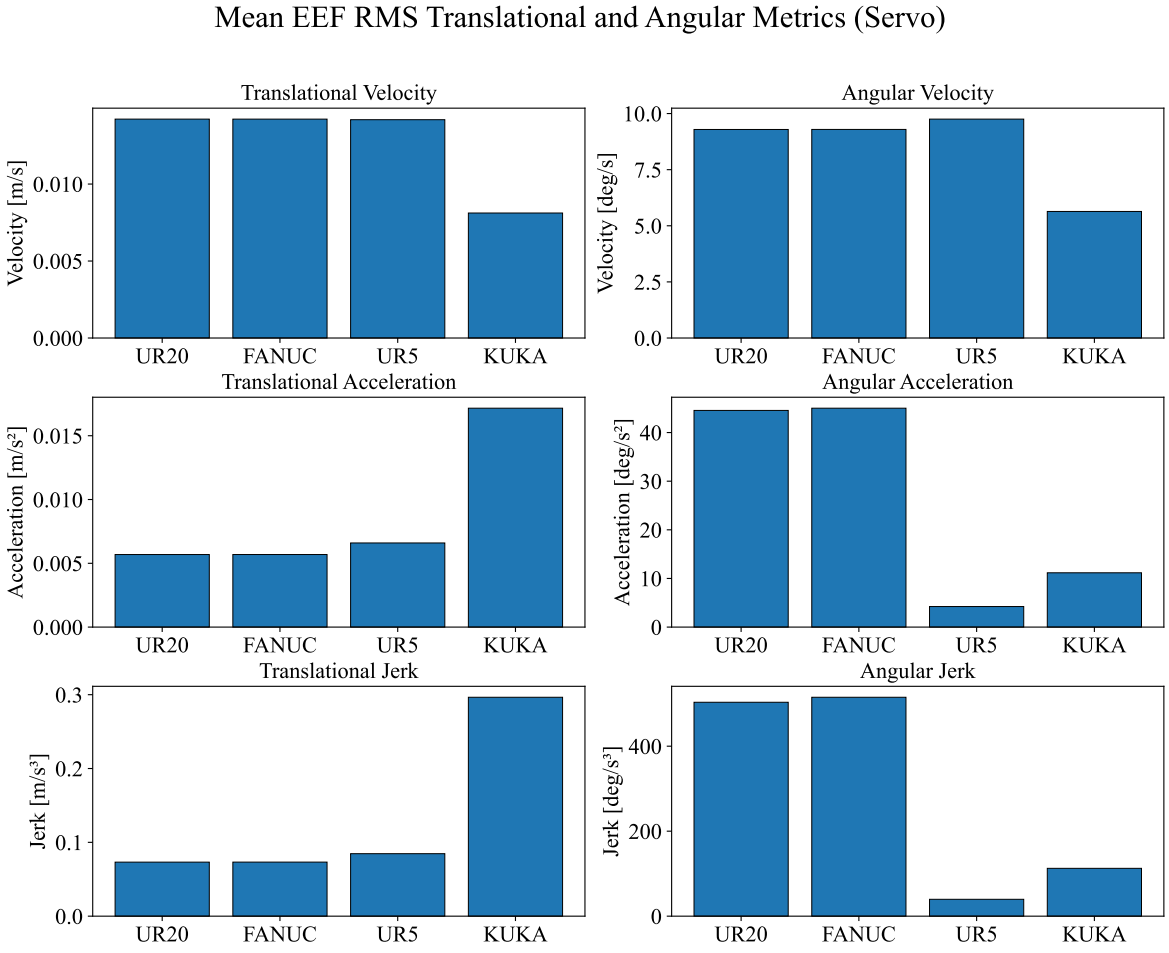


Figure 19. Cartesian space mean RMS metrics for the MoveIt Servo. Cartesian space metrics are divided in translational and angular constituents.

4.4.5 Motion Planner Comparison

Smoothness scores are categorized into joint-space and Cartesian-space metrics. Joint-space smoothness, is presented as an average jerk RMS score of all runs. Cartesian-space smoothness is divided into translational, angular as an average across all robots tested. For quantifying overall smoothness, a unit-consistent velocity-adjusted smoothness both in the Cartesian and Joint-space domains are considered.

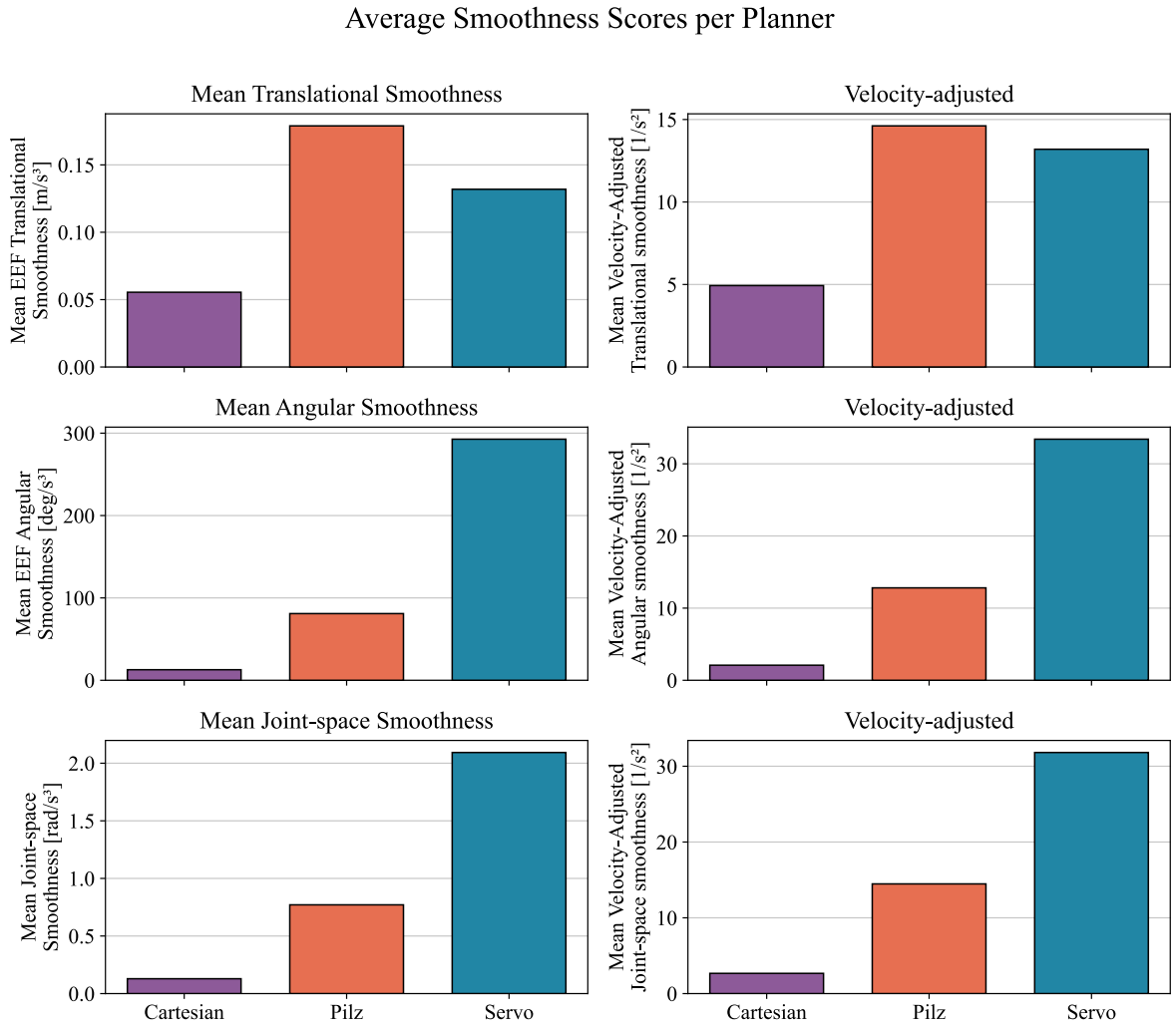


Figure 20. Mean smoothness scores averaged across robots and runs. Lower is smoother. Numerical data in tables 2 and 3

Table 2. Mean RMS jerk/smoothness of each planner tested (lower is smoother).

Metric	**Cartesian**	**Pilz**	**Servo**
Translational RMS jerk [m/s^3]	0.055	0.179	0.132
Angular RMS jerk [deg/s^3]	12.83	80.96	292.7
Joint-space RMS jerk [rad/s^3]	0.129	0.771	2.093

Table 3. Mean RMS velocity-adjusted smoothness per planner (lower is smoother).

Metric (velocity-adjusted)	**Cartesian**	**Pilz**	**Servo**
Translational smoothness [$1/s^2$]	4.93	14.61	13.2
Angular smoothness [$1/s^2$]	2.09	12.81	33.41
Joint-space smoothness [$1/s^2$]	2.67	14.47	31.81

The velocity-adjusted translational, angular and joint-space values in table 3 were averaged to produce table 4.

Table 4. Averaged smoothness scores (lower is better).

Planner	**Avg. Smoothness [$1/s^2$]**
Cartesian	3.23
Pilz	13.96
Servo	26.14

Consistency of Velocity

Consistency of velocity was quantified by calculating an average coefficient of velocity variation each run, defined as the standard velocity deviation divided by the mean velocity. The reported score is the negative base-10 logarithm of the variation coefficient, so higher values indicate more consistent velocities in Cartesian space.

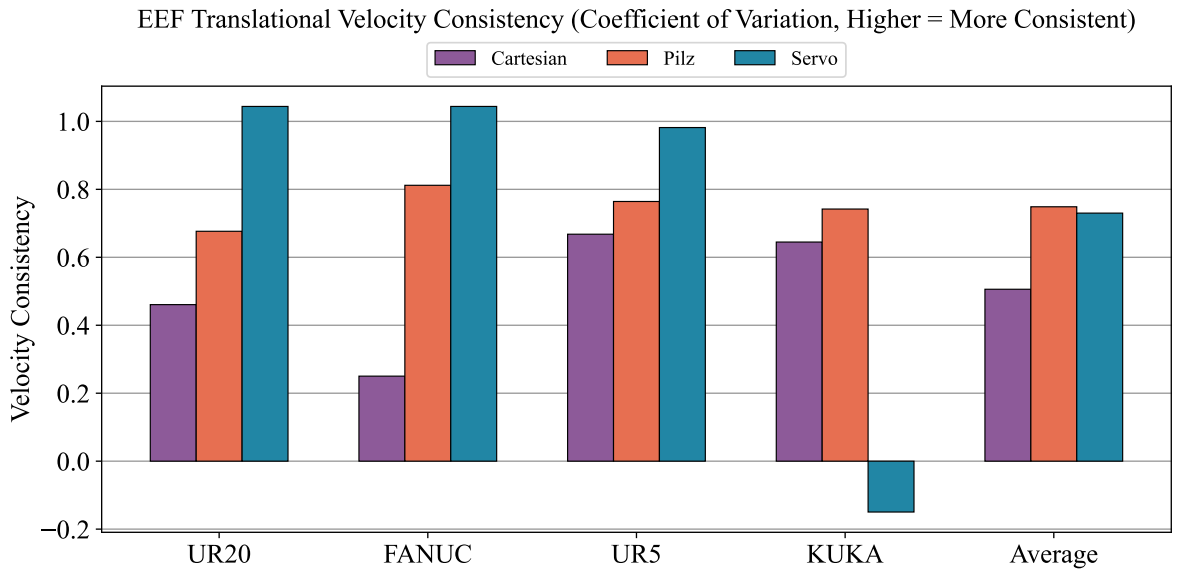


Figure 21. End-effector consistency of velocity as a measure of velocity deviation from the mean velocity. Numerical data in table 5 (higher is better).

Table 5. Velocity Consistency scores (higher is better)

Planner	UR20	FANUC	UR5	KUKA	Average
Cartesian	0.461	0.25	0.668	0.645	0.506
Pilz	0.677	0.812	0.764	0.742	0.749
Servo	1.044	1.044	0.982	-0.15	0.73

The velocity consistency scores are clearly reflected in the time-velocity graphs 22 & 23, where Servo overall has the most consistent velocity, the exception being in the case with Kuka. The reason for the extremely low velocity consistency score in the Kuka trials is that Servo failed to complete any trajectory after trying to force the Kuka robot beyond its joint limits.

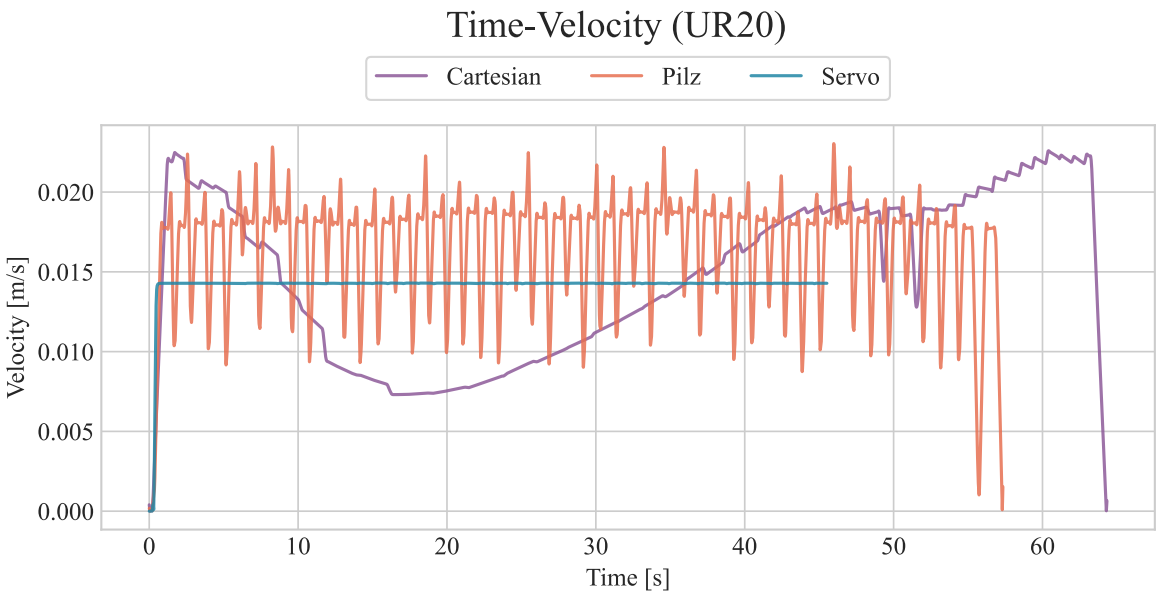


Figure 22. Time-velocity graph of one UR-20 run. All motion planner velocity profiles are plotted.

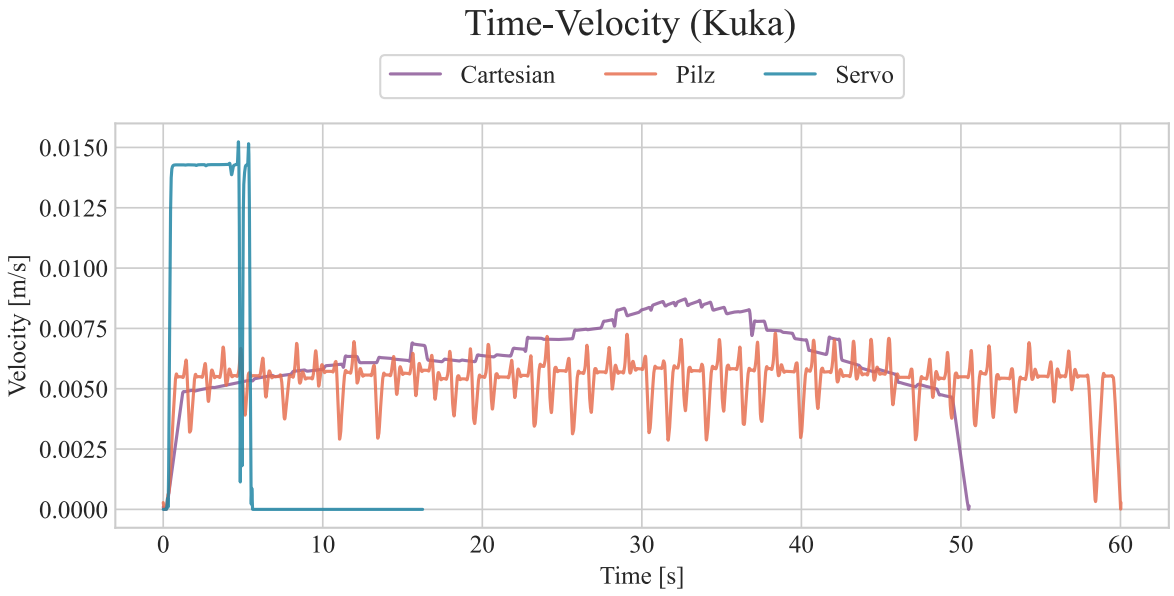


Figure 23. Time-velocity graph of one Kuka run. All motion planner velocity profiles are plotted.

Overall score

A welding or cutting robot should preferably move at a consistent velocity, have minimal jerk and stay as close to its planned path as possible. Therefore a final composite score of performance is also presented. For the final performance score. The averaged results from tables 1, 4, 5 were used. Each score was normalized for the overall scoring. An unweighted score and a score with the weights in table 6 applied is presented in table 7.

Table 6. Overall score metric weights

Metric	Mean RMS pos. err.	Mean RMS ang. err.	Smoothness	Vel. Consistency
Weight	1.0	0.5	0.8	0.8

Table 7. Overall score (higher is better)

Planner	Cartesian	Pilz	Servo
Unweighted score	0.7500	0.8829	0.2305
Weighted score	0.6813	0.5750	0.1844

5. Discussion & Conclusion

The aim of this thesis was to design and develop a robot-agnostic ROS 2-based pipeline for enabling easy trajectory and motion execution aimed at robotic cutting and welding tasks, something that is lacking in the current open-source tool set. The functionality of the solution is demonstrated and the solution has been finalized and published as a ROS-package. Additionally, this thesis has a benchmarking part of MoveIt motion planners on Cartesian-space trajectory tasks, something which has not been clearly documented before.

5.1 Importance of Tool Rotation

Whether a rotation around its own axis negatively affects weld quality is somewhat inconclusive. [22] claims it does not. But since a significant amount of research has been dedicated to improving the quality of robotic welds, for example [23], [24], and some welding torches are not symmetrical, the assumption in this thesis is that it does impact welding quality to a significant degree, making controlling end-effector rotation around its own axis highly relevant.

5.2 Interpreting Results

Given the context of the developed solution it is important to point out that the scoring associated with the trajectory and motion execution back-ends is not a score that is necessarily only characterizing for the actual motion planners in MoveIt. In reality, making one-size-fits all drivers (the ROS-node responsible for calculating the specific parameters required by the motion planning API, and in the case of Servo, actually running a control system), that actually work well, is a hard task. In other words, there is not a clearly discernible way of determining how much of the score is representative of the performance of the motion planner itself or the actual driver node. So while the material presented for this thesis is not enough to draw conclusions about MoveIt's motion planners in industrial robotic cutting and welding tasks, what can be concluded from the results is the following:

- The back-end that implements MoveIt Servo is far superior in terms of velocity consistency, in fact, it displays consistent velocity control across robot models.
- The back-ends that implement Pilz Industrial Motion Planner and MoveIt ComputeCartesianPath are comparable to each other, and far superior to the Servo back-end in terms of executed trajectory to planned path Cartesian error.
- The back-end implementing MoveIt ComputeCartesianPath is superior in jerk smoothness.

- The back-end implementing MoveIt ComputeCartesianPath distributes velocity, acceleration and jerk across the joints of the tested robots to a higher degree than the other planners.

5.3 Weaknesses in Testing & Current Solution

The basis for the results were three runs on one path with a randomized starting seed joint state for each planner and for each robot, the exception being for the runs with MoveIt Servo and Kuka iiwa14, in this case only two runs were completed as they failed to execute the motion plan due to overly strict joint limits and impatience got the best of the author. (As an additional note, it should be stated that Kuka iiwa14 seems like an ill-suited platform to be used for tracing Cartesian paths). In this thesis a variety of paths for each robot and each planner were not tested, and more extensive testing would likely lead to more representative results.

The seed state generator explain in section 3.1.5 almost always ensures successful trajectory generation. (The only planner that might still fail with a good seed state is Servo, but this was only observed with Kuka iiwa14, which has very strict joint limits). For this reason, metrics related to planning failure, and discontinuities were not considered interesting for evaluation.

Computation time on the platform used was consistently under 1 second. In most cases it varied from 10-100 ms. As the target application is not real-time recalculation of trajectories, this was deemed not interesting to evaluate.

5.3.1 Velocity Control

Velocity control is essential when comparing the different motion execution back-ends, as it directly influences accuracy, smoothness and the derived motion characteristics like acceleration and jerk. For example, executing the same trajectory slower will likely result in overall lower accelerations and a smoother motion profile, not necessarily due to better planning, but simply due to reduced speed. In real welding and cutting applications, maintaining a consistent end-effector speed is functionally critical for operations. Hence, an evaluation that fails to consider velocity control may misrepresent a planner's true performance in specific tasks. To counter this issue, the smoothness metric was adjusted for velocity, although it is still not a perfect metric.

Currently, there is no standard way to enforce both lower- and upper-bound velocity constraints for movement execution in MoveIt across different robot models. Upper-bound velocity constraints are generated by the MoveIt Setup Assistant on a robot-by-robot basis, and each respective planner interprets these constraints differently. The planner that currently offers the

most standardized velocity control is Servo, but in practical applications Servo would require fine tuning of output gains based on specific robot kinematics.

Further complicating the issue is the inconsistency in joint limits in MoveIt. As previously mentioned, Upper-bound velocity and acceleration constraints are by default generated by the MoveIt Setup Assistant, and can be manually re-defined in the joint_limits.yaml. The actual quality of this file depends heavily on the source of the URDF description. In the case of every robot model evaluated, a maximum acceleration limit is missing, and a placeholder had to be added. To eliminate variability due to different joint limits, all joint velocity and acceleration limits were manually standardized to 1 m/s and 1 m/s², respectively, between the evaluated robots. Eliminating one source of ambiguity.

5.4 Suggestions for improvements

The driver nodes implemented in the system that interface with the MoveIt planning APIs could be improved to a large degree. Some concrete suggestions are:

- The MoveIt Servo back-end has very weak performance at the moment. The performance of the MoveIt Servo back-end can surely be radically improved just with better code. If anything, the current back-end just serves as proof-of-functionality.
- All back-ends should be improved to allow for much better velocity control. Right now Cartesian and Pilz depend heavily on the density of way-points in the provided path.
- Much more extensive ROS-parameterization. Both for MoveIt motion planner API parameters to eliminate the need for rewriting the source code every time the user wishes to edit the behavior of the motion planner. But also for changing other meta-parameters like velocity on the fly.
- The Pilz driver should be further improved to eliminate the jerky motion observed. This could probably be accomplished simply by tweaking the blending radius of the motion blending planner.

The path projection script is bare-bones. Some functionality is limited to changing the Python source code, which is bad in terms of usability.

Reach integration should possibly be reconsidered. The developed seed-state finding algorithm also accomplishes the task of evaluating path feasibility, and in some ways better than Reach since it considers path continuity.

The seed-state finding algorithm can be improved to also have functionality for caching good seed states when developing trajectories in industrial settings. It could also be improved to consider joint-state motion profiles, not just scoring seed states based on whether they allow for a trajectory to be executed, but how little the joints actually have to move in execution.

5.5 Additional Comments

By no means is the system limited to processing only cylindrical shapes; regardless, for most of the demonstrations, a cylinder is used for relevance in the field of pipe cutting and welding.

The ROS package is available at: https://github.com/carlhjal/Surfmotion.git.

References

[1] Xu Y., Cheng L., Yang J., Ji Y., Wang H., Sun H., Liu C., and Zhang B. Cartesian space track planning for welding robot with inverse solution multi-objective optimization. *Measurement and Control* 55.7 (2022). _eprint: https://doi.org/10.1177/00202940221106570, pp. 583–594. DOI: 10.1177/00202940221106570. https://doi.org/10.1177/00202940221106570.

[2] Siciliano B., Sciavicco L., Villani L., and Oriolo G. Robotics: Modelling, Planning and Control. Springer, 2010. 161 pp.

[3] iStock-609935512.jpg (2329×1287). https://robodk.com/blog/wp-content/uploads/2023/12/iStock-609935512.jpg (05/19/2025).

[4] LinuxCNC. https://linuxcnc.org/ (05/05/2025).

[5] Zhou P., Peng R., Xu M., Wu V., and Navarro-Alarcon D. Path Planning With Automatic Seam Extraction Over Point Cloud Models for Robotic Arc Welding. *IEEE Robotics and Automation Letters* 6.3 (July 2021), pp. 5002–5009. DOI: 10.1109/LRA.2021.3070828. https://ieeexplore.ieee.org/document/9394722/ (05/05/2025).

[6] SintefManufacturing/python-urx. original-date: 2013-03-23T10:33:39Z. May 3, 2025. https://github.com/SintefManufacturing/python-urx (05/05/2025).

[7] ros-industrial-consortium/descartes. original-date: 2014-05-06T23:55:32Z. Apr. 8, 2025. https://github.com/ros-industrial-consortium/descartes (05/04/2025).

[8] ros-industrial/noether. original-date: 2017-05-03T12:27:54Z. Apr. 17, 2025. https://github.com/ros-industrial/noether (05/04/2025).

[9] Diankov R. Automated Construction of Robotic Manipulation Programs. thesis. Carnegie Mellon University, Aug. 26, 2010. DOI: 10.1184/R1/6714905.v1. https://kilthub.cmu.edu/articles/thesis/Automated_Construction_of_Robotic_Manipulation_Programs/6714905/1 (05/05/2025).

[10] The Orocos Project – Smarter control in robotics & automation! https://orocos.org/ (05/19/2025).

[11] Liu G., Sun W., and Li P. Motion capture and AR based programming by demonstration for industrial robots using handheld teaching device. *Scientific Reports* 14.1 (Oct. 6, 2024), p. 23259. DOI: 10.1038/s41598-024-73747-4. https://doi.org/10.1038/s41598-024-73747-4.

[12] Funes-Lora M. A., Vega-Alvarado E., Rivera-Blas R., Calva-Yáñez M. B., and Sepúlveda-Cervantes G. Novel surface optimization for trajectory reconstruction in industrial

robot tasks. *International Journal of Advanced Robotic Systems* 18.6 (Nov. 1, 2021), p. 17298814211064767. DOI: 10.1177/17298814211064767. https://journals.sagepub.com/doi/10.1177/17298814211064767 (05/02/2025).

[13] MoveIt 2 Documentation — MoveIt Documentation: Rolling documentation. https://moveit.picknik.ai/main/index.html (05/05/2025).

[14] ros-industrial/reach. original-date: 2019-07-16T00:18:04Z. Mar. 27, 2025. https://github.com/ros-industrial/reach (05/05/2025).

[15] ros-industrial/reach_ros2. original-date: 2023-05-09T18:23:49Z. Apr. 30, 2025. https://github.com/ros-industrial/reach_ros2 (05/05/2025).

[16] Zhou Q.-Y., Park J., and Koltun V. Open3D: A Modern Library for 3D Data Processing. Jan. 30, 2018. DOI: 10.48550/arXiv.1801.09847. arXiv: 1801.09847[cs]. http://arxiv.org/abs/1801.09847 (05/04/2025).

[17] Turk G. The PLY Polygon File Format (). https://gamma.cs.unc.edu/POWERPLANT/papers/ply.pdf (05/03/2025).

[18] The PCD (Point Cloud Data) file format — Point Cloud Library 1.15.0-dev documentation. https://pointclouds.org/documentation/tutorials/pcd_file_format.html# (05/03/2025).

[19] Bentley J. L. Multidimensional binary search trees used for associative searching. *Communications of the ACM* 18.9 (Sept. 1975), pp. 509–517. DOI: 10.1145/361002.361007. https://dl.acm.org/doi/10.1145/361002.361007 (05/03/2025).

[20] Virtanen P., Gommers R., Oliphant T. E., Haberland M., Reddy T., Cournapeau D., Burovski E., Peterson P., Weckesser W., Bright J., Walt S. J. v. d., Brett M., Wilson J., Millman K. J., Mayorov N., Nelson A. R. J., Jones E., Kern R., Larson E., Carey C. J., Polat İ., Feng Y., Moore E. W., VanderPlas J., Laxalde D., Perktold J., Cimrman R., Henriksen I., Quintero E. A., Harris C. R., Archibald A. M., Ribeiro A. H., Pedregosa F., Mulbregt P. v., and Contributors S. 1. 0. SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17.3 (Mar. 2, 2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2. arXiv: 1907.10121[cs]. http://arxiv.org/abs/1907.10121 (05/03/2025).

[21] Zhang W., Xiao J., Chen H., and Zhang Y. Measurement of three-dimensional welding torch orientation for manual arc welding process. *Measurement Science and Technology* 25.3 (Mar. 1, 2014), p. 035010. DOI: 10.1088/0957-0233/25/3/035010. https://iopscience.iop.org/article/10.1088/0957-0233/25/3/035010 (05/04/2025).

[22] De Maeyer J., Moyaers B., and Demeester E. Cartesian path planning for arc welding robots: Evaluation of the descartes algorithm. *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). ISSN: 1946-0759. Sept. 2017, pp. 1–8. DOI: 10.1109/ETFA.2017.8247616. https://ieeexplore.ieee.org/document/8247616/ (05/04/2025).

[23] Parvez S., Abid M., Nash D. H., Fawad H., and Galloway A. Effect of Torch Angle on Arc Properties and Weld Pool Shape in Stationary GTAW. *Journal of Engineering Mechanics* 139.9 (Sept. 2013), pp. 1268–1277. DOI: 10.1061/(ASCE)EM.1943-7889.0000553. https://ascelibrary.org/doi/10.1061/%28ASCE%29EM.1943-7889.0000553 (05/04/2025).

[24] Lu H., Wu Z., Zhang Y., Wang Y., Liu S., Huang H., Liu M., and Liu S. Towards a Uniform Welding Quality: A Novel Weaving Welding Control Algorithm Based on Constant Heat Input. *Materials* 15.11 (Jan. 2022). Number: 11 Publisher: Multidisciplinary Digital Publishing Institute, p. 3796. DOI: 10.3390/ma15113796. https://www.mdpi.com/1996-1944/15/11/3796 (05/04/2025).

License

I, Carl Hjalmar Love Hult

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   SurfMotion: An Open Source Pipeline for Robotic Pipe Cutting and Welding

   supervised by Karl Kruusamäe

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Carl Hjalmar Love Hult

20.05.2025