The Potential of Constraint-Based Robot Programming for Welding Robots in the Norwegian Industry

Vebjørn Bergsholm Bjørhovde

2020-12-17

Preface

First I want to give thanks to my supervisor Lars Tingelstad for very useful guidance into the field of constraint-based programming, and general guidance throughout the project.

Thanks to Adam Leon Kleppe for introduction and help at the robotics lab.

My gratitude to Bassam Hussein for useful input from on how to best conduct an interview and the necessary precautions to avoid breaking any laws regarding the collection of personal data. I would also like to thank the anonymous interview subjects who gave their time to answer our questions.

Last but not least I want to thank Thea Holmedal for our useful discussions and cooperation while working on this report.

Vebjørn B. Bjørhovde

Summary

This project investigates the possibility of implementing constraint-based robot programming to welding robots in the Norwegian industry. It is also preparatory work for a master's thesis.

Current welding methods and sensors are presented along with an overview of common welding robots and robot programming methods. A literature review on state of the art frameworks for constraint-based robot programming is also conducted, with a detailed presentation of expressiongraph-based Task Specification (eTaSL)/expressiongraph-based Task Controller (eTC), Stack of Tasks (SoT) and instantaneous Task Specification using Constraints (iTaSC). In addition to the literature review, interviews were conducted to map the current state for welding robots in the Norwegian industry.

Based on the gathered information, the challenges and advantages of constraint-based robot programming for welding robots in the Norwegian industry are discussed. Briefly summarised, constraint-based robot programming could lead to more flexible robots, higher degree of automation and easier robot programming of complex welding tasks.

In addition to this discussion, a robot cell in the Department of Mechanical and Industrial Engineering (MTP) at the Norwegian University of Science and Technology (NTNU) is presented. This cell can be used for further work on applying constraint-based robot programming to welding robot. Some preparatory work was done in this project, a simplified model of the robot was made along with code for a simple visualiser of the robot.

Sammendrag

Denne prosjektoppgaven undersøker mulighetene for å implementere begrensningsbasert robotprogrammering for sveiseroboter i den norske industrien. Oppgaven er også forberedende arbeid for en masteroppgave.

Dagens sveisemetoder og sensorer blir først presentert, sammen med et utvalg av typiske sveiseroboter. Deretter presenteres et litteratursøk av den nyeste forskningen innen rammeverk for begrensningsbasert robotprogrammering. Tre rammeverk blir presentert i detalj: "Expressiongraph-based Task Specification" (eTaSL)/"expressiongraph-based Task Controller" (eTC), "Stack of Tasks" (SoT) og "instantaneous Task Specification using Constraints" (iTaSC). I tillegg til litteratursøket har det blitt gjennomført intervjuer for å kartlegge hvordan robootsveising brukes i den norske industrien i dag.

Med utgangspunkt i informasjonen fra intervjuene og litteratursøket blir fordelene og utfordringene med begrensningsbasert robotprogrammering i den norske industrien diskutert. Kort oppsummert kan begresningsbasert robotprogrammering lede til mer fleksible roboter, høyere grad av automatisering og lettere robotprogrammering for komplekse sveiseoperasjoner.

I tillegg til denne diskusjonen presenteres en robotcelle fra Institutt for maskinteknikk og produksjon (MTP) ved Norges teknisk-naturvitenskapelige universitet (NTNU). Denne cellen kan brukes for videre arbeid med begrensningsbasert robotprogrammering av sveiseroboter. Noe forberedende arbeid er gjort. En forenklet modell av roboten er laget sammen med en kode for enkel visualisering av roboten.

Contents

List of Figures

List of Tables

Chapter 1.

Introduction

Constraint-based robot programming is a way of programming robots where constraints are defined and translated into an end-effector trajectory by a solver. Constraints can originate from the robot hardware, the task it is set to do or the surrounding environment. This way of programming can allow for more intuitive robot programming as well as faster and easier programming of complex tasks.

1.1. Problem Statement

Constraint-based robot programming is mainly used in research environments and is yet to be implemented in the industry. There is little research on the field of implementing constraint-based robot programming for welding robots, and the benefits of such an implementation are currently unclear.

With a focus on the Norwegian industry, this report aims to explore the possibilities of introducing constraint-based robot programming in robotic welding. A literature review on state of the art constraint-based robot programming frameworks and welding technologies will be conducted. In addition to this the current situation for robotic welding in the Norwegian industry will mapped through interviews. This work results in a discussion regarding advantages and challenges in constraint-based programming for welding robots in the Norwegian industry.

Additionally this report aims to be preparatory work for a master's thesis to be written on the subject. As preparations a robot cell set up for further work will be presented along with other preparatory work.

1.2. Methodology

To give an overview on welding methods, sensors and previous work on constraint-based robot programming, a literature review was conducted. This was considered the most available and efficient way of gathering information on the topic. Source criticism was utilised to not include false or outdated information, which is always a risk when doing a literature research.

As for investigating robotic welding in the Norwegian industry, interviews were regarded as the best solution. Different companies working with robotic welding in Norway was contacted, and the interview templates seen in Appendix A was presented. After having the interview subject read through the questions, the interviews were completed by phone. Interviews was chosen as the amount of literature on robotic welding in the industry is mostly outdated. While interviews yields hands on information, it is important to be critical to conclusions based off interviews. They can be misleading if the subjects fails to represent the whole target group. In this report the subjects were limited to Norwegian industries due to time considerations. Note that the interviews where conducted in cooperation with Thea Holmedal, so the interview results will occur in another report as well.

1.3. Structure of the Report

The structure of the report is inspired by the IMRaD structure for scientific writing where a text is divided in Introduction, Material and Method, Results and Discussion. For this report the method is merged with the introduction. Figure 1.1 shows the structure of the report.

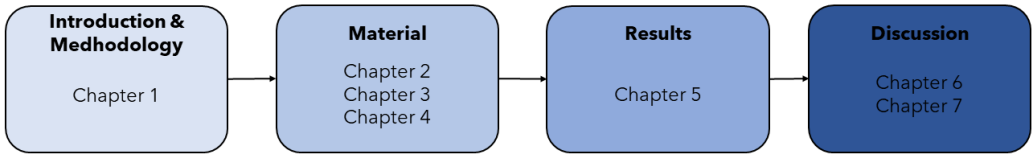


Figure 1.1.: A visual representation of the report structure inspired by IMRaD (Introduction, Method and Material, Results and Discussion).

Chapter 2.

Robot Kinematics

This chapter will summarise some important aspects of robot kinematics for open chain manipulators. This is a robot where one end is not fastened, like the robotic arms commonly seen in the industry. The theory is meant to give a better understanding of the work that will be presented later in the report. All sections in this chapter are based off the work done in [18] unless stated otherwise. All equations presented are also taken from [18].

2.1. Degrees of Freedom

Degrees of freedom (DOF) defines the range of motion for an object. For this report, 3-dimensional space will be considered. A rigid body in space has a configuration, which is a description of the body position and orientation. The number of parameters needed to fully describe the body is equal to its number of DOF. To illustrate this concept, consider the cube shown in Figure 2.1. As the cube is not constrained in any way, it can be moved in the x, y and z-directions. It can also be rotated by an angle γ, β and α around the x, y and z-axis respectively. For rigid bodies it holds that:

$$\text{DOF} = \text{(sum of freedoms of the bodies)-(number of independent constraints)}$$

From this it can be concluded that a rigid body in space with no constraints has six DOF. The next step is to define the DOF for a robot based on the type and number of joints it has.

In robotics there are different types of joints with different DOF. The most common in industrial robots is the revolute joint, which yields one DOF as it rotates about one axis. The DOF for a robot is calculated by Grübler's formula:

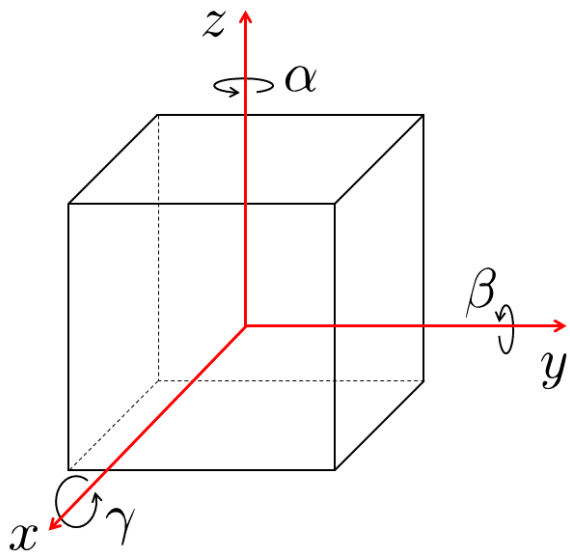


Figure 2.1.: Degrees of freedom (DOF) shown for a cube in 3D.

$$\text{DOF} = m(N - 1 - J) + \sum_{i=1}^{J} f_i \tag{2.1}$$

Where m is the number of DOF for a rigid body, N is the number of robot links including the ground it is mounted on, J is the number of joints and f_i is the DOF for joint i. Note that for a robot with only revolute joints, the DOF is equal to the number of joints. From (2.1) it is seen that a robot can exceed the six DOF needed for free movement in 3D, this will cause redundancy which is the topic of the next section.

2.2. Redundancy

Redundancy has been defined differently by several authors in literature. Several definitions were presented in [7], and a common definition was suggested. Parts of this definition will be summarised in this section. Two definitions are needed before defining redundancy. The workspace of a robot is defined as the space a robot can reach with its end-effector, and the task space is defined as the space where a robot task can be expressed naturally.

The redundancy definition in [7] considers industrial robots with joints that yield one DOF. Let the number of joints in a robot be denoted n and the dimension of the robot's work- and taskspace denoted w and t respectively. With these

parameters, three cases are worth noting:

- $n = w$: This is the normal case where most industrial robots operate today.
- $n > w$: This is a case of kinematic redundancy, where the number of robot joints exceeds the requirement for the robot to operate in the workspace. Kinematic redundancy will make the robot more flexible as it can move more joints without changing the end-effector pose.
- $n > t$: This case is similar to the previous one, but defined as task redundancy. In this case a robot has more DOF than is required for its given task.

In the two latter cases redundancy can be utilised to move the robot while still maintaining the same end-effector pose. This can be used to avoid external obstacles, avoid singularities (Section 2.9), minimise energy consumption, etc.

2.3. Rotation Matrices

Before presenting rotation matrices, the concept of frames must be defined. A frame is simply a coordinate system that can be placed freely in space. Usually a space-frame is set and used as reference when working with robotics. A body-frame is defined as a frame that is always coincident with a frame attached to a body. For practical reasons frames in this report are always right-handed. With these concepts in place, rotation matrices can be explained.

Rotation matrices are used to represent the orientation of a frame, change the reference frame for a vector or frame and rotate a vector or frame. In 3D these matrices are represented by the special orthogonal group $SO(3)$. These 3×3 real matrices $\boldsymbol{R}$ satisfy the following conditions:

$$\begin{aligned} \boldsymbol{R}^T \boldsymbol{R} &= \boldsymbol{I} \\ \det(\boldsymbol{R}) &= 1 \end{aligned}$$

where $\boldsymbol{I}$ is the 3×3 identity matrix. To illustrate the use of rotation matrices Figure 2.2 from [18] will be used, where three different frames are shown along with a point p. The first use of rotation matrices is representing orientation. Orientation is always represented with regards to a reference frame, so imagine a frame {s} coincident with frame {a}. The notation $\boldsymbol{R}_{ij}$ represents the orientation of frame $\{j\}$ with respect to frame $\{i\}$. This yields the following orientations for the three frames:

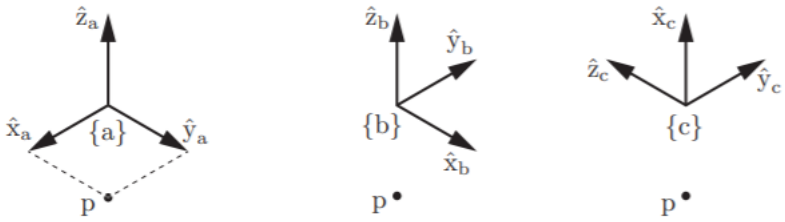


Figure 2.2.: Three coordinate frames with a point p used to demonstrate the use of rotation matrices. Figure from [18].

$$\boldsymbol{R}_{sa} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{R}_{sb} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{R}_{sc} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}$$

The point p is represented as a vector in each frame, note that $\boldsymbol{p}_i$ denotes the point p with reference frame $\{i\}$. This yields the following representations:

$$\boldsymbol{p}_a = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \quad \boldsymbol{p}_b = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}, \quad \boldsymbol{p}_c = \begin{bmatrix} 0 \\ -1 \\ -1 \end{bmatrix}$$

The second use of a rotation matrix is to change the reference frame of another frame or vector. Consider the matrix $\boldsymbol{R}_{sa}$. How can the orientation of frame {a} be represented with respect to frame {b}? This can be done with the following matrix multiplication:

$$\boldsymbol{R}_{ba} = \boldsymbol{R}_{bs}\boldsymbol{R}_{sa} \tag{2.2}$$

Note that $\boldsymbol{R}_{bs} = \boldsymbol{R}_{sb}^T = \boldsymbol{R}_{sb}^{-1}$. The reference frame for vectors can be changed in a similar manner:

$$\boldsymbol{p}_b = \boldsymbol{R}_{ba}\boldsymbol{p}_a \tag{2.3}$$

The last use of rotation matrices is the rotation of a frame or vector. A pure rotation around the unit $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}$ and $\hat{\boldsymbol{z}}$-axis can be represented by (2.4), (2.5) and (2.6) respectively, where θ is the rotation angle about the axis:

$$\mathbf{Rot}(\hat{\boldsymbol{x}}, \theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{2.4}$$

$$\mathbf{Rot}(\hat{\boldsymbol{y}}, \theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.5}$$

$$\mathbf{Rot}(\hat{\boldsymbol{z}}, \theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

When rotating a frame with a rotation matrix, it is done by either post- or premultiplying the rotation matrix $\boldsymbol{R}$ with the orientation of the frame. Premultiplication, $\boldsymbol{R}\boldsymbol{R}_{ij}$, correlates to rotating $\boldsymbol{R}_{ij}$ with respect to frame {i}. Postmultiplication, $\boldsymbol{R}_{ij}\boldsymbol{R}$, correlates to rotating $\boldsymbol{R}_{ij}$ with respect to frame {j}. A vector must always be rotated with respect to its reference frame and is therefore always premultiplied with the rotation matrix.

2.3.1. Exponential Representation of Rotation

Imagine that instead of representing rotations as pure rotations about each coordinate axis, the rotation can be represented by a rotation θ about one arbitrary unit axis $\hat{\boldsymbol{\omega}}$. The resulting rotation matrix can be found by Rodrigues' formula:

$$\mathrm{Rot}(\hat{\boldsymbol{\omega}}, \theta) = e^{[\hat{\omega}]\theta} = \boldsymbol{I} + \sin(\theta)[\hat{\boldsymbol{\omega}}] + (1 - \cos(\theta))[\hat{\boldsymbol{\omega}}]^2 \tag{2.7}$$

where $[\hat{\boldsymbol{\omega}}]$ is the skew-symmetric representation of $\hat{\boldsymbol{\omega}} = [\omega_1, \omega_2, \omega_3]^T$ defined as:

$$[\hat{\boldsymbol{\omega}}] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{2.8}$$

2.3.2. Matrix Logarithm of a Rotation Matrix

In (2.7) a rotation matrix $\boldsymbol{R}$ was found from an arbitrary unit rotation axis $\hat{\boldsymbol{\omega}}$ and a rotation angle θ. The matrix logarithm is the opposite operation where the axis and angle is found from the matrix. The two essential equations used to calculate the matrix logarithm form a rotation matrix $\boldsymbol{R}$ are:

$$\mathrm{tr}(\boldsymbol{R}) = 1 + 2\cos(\theta) \tag{2.9}$$

$$[\hat{\boldsymbol{\omega}}] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} = \frac{1}{2\sin(\theta)}(\boldsymbol{R} - \boldsymbol{R}^T) \tag{2.10}$$

From (2.9) it is possible to find the rotation angle θ, which in turn can be inserted in (2.10) to find the unit rotation axis $\hat{\boldsymbol{\omega}}$. Two special cases are worth noting. When $\boldsymbol{R} = \boldsymbol{I}$, $\theta = 0$ which in turn makes $\hat{\boldsymbol{\omega}}$ undefined. When $\mathrm{tr}(\boldsymbol{R}) = -1$, $\theta = \pi$ and $\hat{\boldsymbol{\omega}}$ can be set to any of the following solutions:

$$\hat{\boldsymbol{\omega}} = \frac{1}{\sqrt{2(1 + r_{33})}} \begin{bmatrix} r_{13} \\ r_{23} \\ 1 + r_{33} \end{bmatrix} \tag{2.11}$$

$$\hat{\boldsymbol{\omega}} = \frac{1}{\sqrt{2(1 + r_{22})}} \begin{bmatrix} r_{12} \\ 1 + r_{22} \\ r_{32} \end{bmatrix} \tag{2.12}$$

$$\hat{\boldsymbol{\omega}} = \frac{1}{\sqrt{2(1 + r_{11})}} \begin{bmatrix} 1 + r_{11} \\ r_{21} \\ r_{31} \end{bmatrix} \tag{2.13}$$

where r_{ij} are the elements of the rotation matrix $\boldsymbol{R}$.

2.4. Transformation Matrices

To describe translation as well as rotation in 3D, the homogeneous transformation matrices are used. These matrices form the special Euclidean group $SE(3)$, and are 4×4 real matrices. They include a rotational matrix $\boldsymbol{R} \in SO(3)$ as well as a column vector $\boldsymbol{p} \in \mathbb{R}^3$ representing translation along the three axes of a frame. Homogeneous transformation matrices $\boldsymbol{T}$ take the form:

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{p} \\ 0 & 1 \end{bmatrix} \tag{2.14}$$

The use of transformation matrices are analogous to that of rotation matrices, but also include position and translation of frames. Transformation matrices

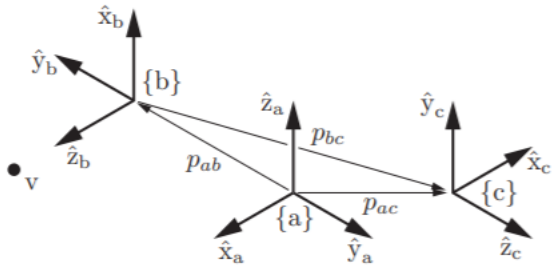


Figure 2.3.: Three frames with a point v used to describe homogeneous transformation matrices. Figure from [18].

can be used to represent the position and orientation, i.e. configuration, of a frame, change the reference frame of a vector or frame or change configuration for a vector or frame. To illustrate the different uses of transformation matrices, consider Figure 2.3 taken from [18]. Imagine once again a reference frame {s} that is coincident with frame {a}.

By using transformation matrices as representations of frame configuration, the following matrices are obtained:

$$\boldsymbol{T}_{sa} = \begin{bmatrix} \boldsymbol{R}_{sa} & \boldsymbol{p}_a \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{T}_{sb} = \begin{bmatrix} \boldsymbol{R}_{sb} & \boldsymbol{p}_b \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & -2 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{T}_{sc} = \begin{bmatrix} \boldsymbol{R}_{sc} & \boldsymbol{p}_c \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The notation is equal to the one for rotation matrices so $\boldsymbol{T}_{ij}$ is the configuration of frame {i} with respect to {j}. The point v is represented by a homogeneous vector $\in \mathbb{R}^4$. In this case the three first elements of the vector are analogous to the case seen for p in Section 2.3, and the last element is equal to one. The rule for changing the reference frame of a transformation matrix is also equal to the one for rotation matrices. Changing the reference frame of $\boldsymbol{T}_{sa}$ from {s} to {b}

can be done in the following manner:

$$\boldsymbol{T}_{ba} = \boldsymbol{T}_{bs}\boldsymbol{T}_{sa} \tag{2.15}$$

Note that $\boldsymbol{T}_{bs} = \boldsymbol{T}_{sb}^{-1}$, where the inverse of a transformation matrix is found as:

$$\boldsymbol{T}^{-1} = \begin{bmatrix} \boldsymbol{R}^T & -\boldsymbol{R}\boldsymbol{p} \\ 0 & 1 \end{bmatrix} \tag{2.16}$$

The rules for changing the configuration of a vector or frame are similar to the rules for rotation matrices. The difference between the two being the translation. When pre-multiplying with a transformation matrix the frame is first rotated with respect to the space-frame according to the rotation matrix $\boldsymbol{R}$, before it is translated with respect to the space-frame according to the vector $\boldsymbol{p}$. When post-multiplying by a transformation matrix the frame is first translated with respect to the body frame, before it is rotated with respect to the body frame. As vectors are only related to one frame they can only be premultiplied to change configuration.

2.5. Twists and Screws

Twists are representations of both the linear and angular velocity of a moving frame. It is a six-dimensional vector consisting of both velocities and can be represented in regards to the fixed frame {s} or the moving body frame {b}. These are the body twist and spacial twist respectively, and are written on the following form:

$$\boldsymbol{\mathcal{V}}_s = \begin{bmatrix} \boldsymbol{\omega}_s \\ \boldsymbol{v}_s \end{bmatrix}, \ \boldsymbol{\mathcal{V}}_b = \begin{bmatrix} \boldsymbol{\omega}_b \\ \boldsymbol{v}_b \end{bmatrix} \in \mathbb{R}^6$$

where $\boldsymbol{\omega}_s$ and $\boldsymbol{\omega}_b$ represents the angular velocity of the frame with respect to the fixed frame and the body frame respectively. $\boldsymbol{v}_b$ represents the linear velocity of the body frame, but $\boldsymbol{v}_s$ does not simply represent the velocity of a frame with respect to {s}.

To explain the physical interpretation of $\boldsymbol{v}_s$ it is useful to look at Figure 2.4 taken from [18]. The body containing both the body frame {b} and the fixed frame {s} is moving with a body twist $\boldsymbol{\mathcal{V}}_b$. Now imagine a point that is attached to the body and coincident with the origin of {s}. This point will have the linear velocity $\boldsymbol{v}_s$, which is a combination of the rotational and linear velocity of the body.

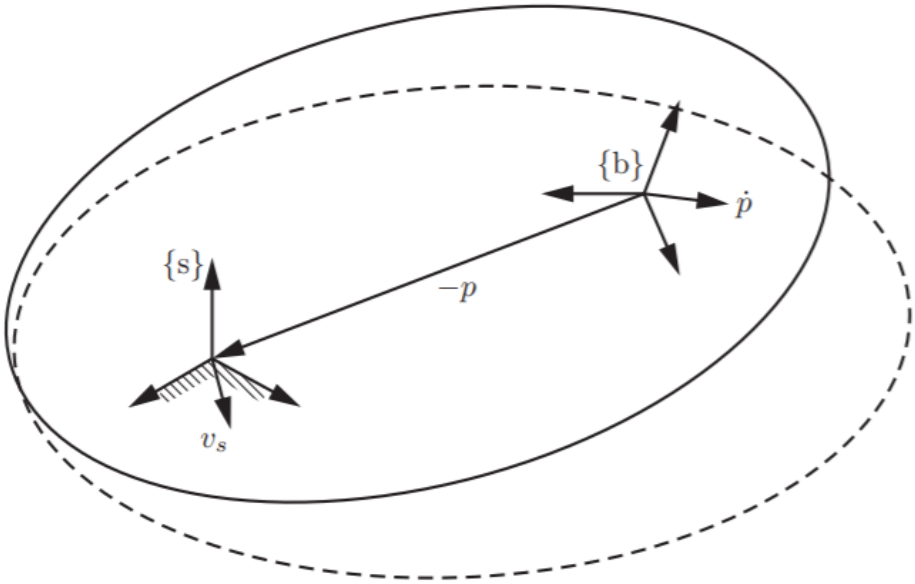


Figure 2.4.: A visual representation of the linear velocity of a twist represented in the fixed space frame {s}. Figure from [18].

A twist can be represented by a normalised screw axis $\boldsymbol{\mathcal{S}}$ multiplied by an angular velocity $\dot{\theta}$. The motion of a frame along screw axis will then replicate the threads of a screw with a a translation along the axis happening simultaneously with a rotation about the axis. The angular velocity around this axis is represented by $\dot{\theta}$. The process of finding a screw axis $\boldsymbol{\mathcal{S}}$ and an angular velocity $\dot{\theta}$ from a twist $\boldsymbol{\mathcal{V}} = \begin{bmatrix} \boldsymbol{\omega}^T & \boldsymbol{v}^T \end{bmatrix}^T$ will now be presented.

If $\boldsymbol{\omega} \neq 0$ then the screw axis $\boldsymbol{\mathcal{S}}$ is equal to the twist normalised by $||\boldsymbol{\omega}||$. The rotational velocity $\dot{\theta}$ is then equal to $||\boldsymbol{\omega}||$. If $\boldsymbol{\omega} = 0$ there is no angular velocity and the twist is just a linear translation. The screw axis $\boldsymbol{\mathcal{S}}$ is then represented by the twist normalised by $||\boldsymbol{v}||$. The velocity $\dot{\theta}$ is then equal to $||\boldsymbol{v}||$. For both cases the twist $\boldsymbol{\mathcal{V}} = \boldsymbol{\mathcal{S}}\dot{\theta}$.

A more formal definition of the screw axis can be written as:

$$\boldsymbol{\mathcal{S}} = \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v} \end{bmatrix} \in \mathbb{R}^6 \tag{2.17}$$

If $\omega = 0$ then $||\boldsymbol{v}|| = 1$ and the twist will be a pure translation along the axis defined by $\boldsymbol{v}$. When $||\boldsymbol{\omega}|| = 1$ the velocity $\boldsymbol{v} = -\boldsymbol{\omega} \times \boldsymbol{q} + h\boldsymbol{\omega}$ where $\boldsymbol{q}$ is a point

on the screw axis and h is the pitch of the screw defined as $\frac{\boldsymbol{\omega}^T \boldsymbol{v}}{\dot{\theta}}$.

It can be useful to express the screw axis $\boldsymbol{\mathcal{S}}$ in matrix form. This expression will also hold for a twist $\boldsymbol{\mathcal{V}}$ as $\boldsymbol{\mathcal{S}}$ is a normalised twist.

$$[\boldsymbol{\mathcal{S}}] = \begin{bmatrix} [\boldsymbol{\omega}] & \boldsymbol{v} \\ 0 & 0 \end{bmatrix} \tag{2.18}$$

2.5.1. Adjoint representation

It is useful to be able to change the reference frame of a twist. This can be done with the adjoint representation of a transformation matrix $\boldsymbol{T}$ which is defined as:

$$[\mathbf{Ad}_T] = \begin{bmatrix} \boldsymbol{R} & 0 \\ [\boldsymbol{p}]\boldsymbol{R} & \boldsymbol{R} \end{bmatrix} \in \mathbb{R}^{6\times 6} \tag{2.19}$$

where $\boldsymbol{R}$ and $\boldsymbol{p}$ are the rotation matrix and translation vector associated with $\boldsymbol{T}$ respectively. The reference frame can be changed from the body frame to the fixed frame by the following equation:

$$\boldsymbol{\mathcal{V}}_s = [Ad_{\boldsymbol{T}_{sb}}]\boldsymbol{\mathcal{V}}_b = Ad_{\boldsymbol{T}_{sb}}(\boldsymbol{\mathcal{V}}_b) \tag{2.20}$$

Note the alternative notation of the adjoint representation multiplied by a twist, this will be used in later sections. Equation 2.20 is also applicable to screw axes.

2.6. Exponential Representation and Matrix Logarithm of Transformation Matrices

Just as rotation matrices can be represented by a rotation θ around an arbitrary unit rotation axis $\hat{\boldsymbol{\omega}}$, homogeneous transformation matrices can be represented by a screw axis $\boldsymbol{\mathcal{S}}$ and a distance θ along the screw axis. To transform this representation into a homogeneous transformation matrix in SE(3), the matrix exponential is utilised. It is defined as:

$$e^{[\boldsymbol{\mathcal{S}}]\boldsymbol{\theta}} = \begin{bmatrix} e^{[\hat{\omega}]\theta} & (\boldsymbol{I}\theta + (1-\cos\theta)[\hat{\boldsymbol{\omega}}] + (\theta - \sin\theta)[v]^2)\boldsymbol{v} \\ 0 & 1 \end{bmatrix} \tag{2.21}$$

Analogous to rotation matrices, it is also possible to find a screw axis $\boldsymbol{S}$ and a distance θ from a transformation matrix $\boldsymbol{T}$ by using the matrix logarithm operation.

Given a transformation matrix on the form shown in (2.14). The rotational axis $\hat{\boldsymbol{\omega}}$ and rotation angle θ can be found from the matrix logarithm of $\boldsymbol{R}$ as described in Section 2.3.2. The linear velocity $\boldsymbol{v}$ is found from the following equation:

$$\boldsymbol{v} = (\frac{1}{\theta}\boldsymbol{I} - \frac{1}{2}[\hat{\boldsymbol{\omega}}] + (\frac{1}{\theta} - \frac{1}{2}\cot(\frac{\theta}{2}))[\hat{\boldsymbol{\omega}}]^2)\boldsymbol{p}$$

For the matrix logarithm of $\boldsymbol{T}$ there is one special case to consider. If $\boldsymbol{R} = \boldsymbol{I}$ there is no rotation and the motion is purely transnational. If that is the case set $\hat{\boldsymbol{\omega}} = \boldsymbol{0}$, $\boldsymbol{v} = \frac{\boldsymbol{p}}{||\boldsymbol{p}||}$ and $\theta = ||\boldsymbol{p}||$.

2.7. Forward Kinematics

Forward kinematics in robotics is the calculation of the end-effector configuration based on the joint angles of the robot. There are different ways of calculating forward kinematics, a widespread method uses Denavit-Hartenberg parameters. In this method each link of the robot is given a frame, and the forward kinematics of the robot can be calculated based on the relation between each frame. The method that will be explained in this section is the product of exponential (PoE) formula, as this method correlates well with the previously explained theory and is the method used in [18].

PoE is based off the exponential representation of homogeneous transformation matrices and their product. Consider the open chain manipulator with n rotational joints shown in Figure 2.5 taken from [18]. Define the fixed space frame {s} and an end-effector frame {b}. Also define a homogeneous transformation matrix $\boldsymbol{M} = \boldsymbol{T}_{sb}(\boldsymbol{0})$ representing the end-effector configuration with all n joints angles equal to zero. This is the zero-position or home-position.

PoE can now be divided into two different approaches depending on which frame is used as reference for defining the exponential representations. This section will focus on the space-frame formulation where {s} is used as reference. After this explanation a brief summary of the calculations using the end-effector frame {b} as reference will be given.

Each of the n rotational joints on the robot can be defined by a screw axis $\boldsymbol{\mathcal{S}}_i$ where $i = 1, 2, ..., n$. To calculate the axes see (2.17). The upper part of $\boldsymbol{\mathcal{S}}_i$ is a vector representing the rotational axis of joint i. As the motion around the joints are purely rotational, the pitch h is zero. This simplifies the velocity calculation and we can write the screw axes for each joint of the robot as:

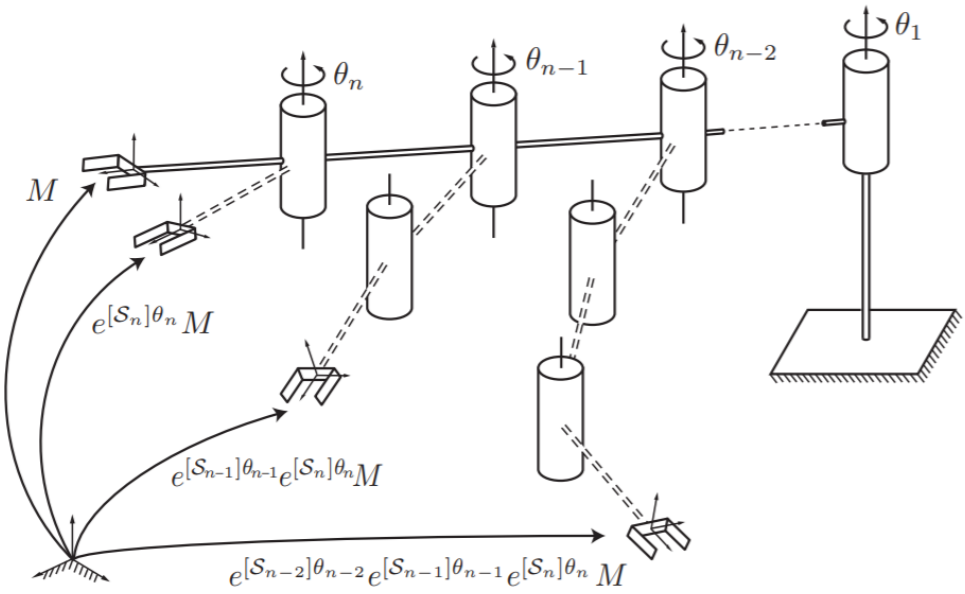


Figure 2.5.: An n-link open chain manipulator used to explain the product of exponentials (PoE) formula. Figure from [18].

$$\boldsymbol{\mathcal{S}}_i = \begin{bmatrix} \boldsymbol{\omega}_i \\ -\boldsymbol{\omega}_i \times \boldsymbol{q}_i \end{bmatrix} \quad i = 1, 2, ..., n \tag{2.22}$$

where $\boldsymbol{q}_i$ is a point on the rotational axis of joint i. After finding a screw axis for each joint, they can be represented in exponential form by using (2.21). The value θ_i is the joint angle of joint i. With the zero-position and an expression for each joint angle it is now possible to calculate the end-effector configuration $\boldsymbol{T}_{sb}(\boldsymbol{\theta})$ by pre-multiplying the chain of exponential representations with the zero-position:

$$\boldsymbol{T}_{sb}(\boldsymbol{\theta}) = e^{[\boldsymbol{\mathcal{S}}_1]\theta_1} e^{[\boldsymbol{\mathcal{S}}_2]\theta_2} ... e^{[\boldsymbol{\mathcal{S}}_n]\theta_n} \boldsymbol{M} \tag{2.23}$$

Where $\boldsymbol{\theta}$ is a vector containing all joint angles $\theta_1, \theta_2 ... \theta_n$.

Calculating the forward kinematics using the end-effector frame {b} is similar to using the space-frame {s}. There are two differences in the approach, the first one being how to calculate the screw axes. These are now calculated with respect to the end-effector frame instead of the base frame and denoted $\boldsymbol{\mathcal{B}}_i$. The second difference is that the zero-position is post-multiplied by the chain of exponential representations instead of pre-multiplied. This yields the following equation for

finding the end-effector pose $\boldsymbol{T}$:

$$\boldsymbol{T}_{sb}(\boldsymbol{\theta}) = \boldsymbol{M}e^{[\boldsymbol{\mathcal{B}}_1]\theta_1}e^{[\boldsymbol{\mathcal{B}}_2]\theta_2}...e^{[\boldsymbol{\mathcal{B}}_n]\theta_n} \tag{2.24}$$

2.8. The Jacobian

For an open chain manipulator, the Jacobian represents the relationship between the velocity of the end-effector and the joint velocity such that:

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \tag{2.25}$$

where $\dot{\boldsymbol{x}} \in \mathbb{R}^m$ is the end-effector velocity, $\boldsymbol{\theta} \in \mathbb{R}^n$ is a list of current joint angles, $\dot{\boldsymbol{\theta}}$ is a list of joint velocities and $\boldsymbol{J}(\boldsymbol{\theta}) \in \mathbb{R}^{m\times n}$ is the Jacobian matrix.

The two types of Jacobians that are relevant for this section are the space Jacobian and the body Jacobian. The space Jacobian $\boldsymbol{J}_s$ correlates the spacial twist of the end-effector to the joint velocities and is defined for a robot manipulator with n joints as:

$$\boldsymbol{\mathcal{V}}_s = \boldsymbol{J}_s(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \tag{2.26}$$

where the ith column of $\boldsymbol{J}_s(\boldsymbol{\theta})$ is:

$$\boldsymbol{J}_{si}(\boldsymbol{\theta}) = \mathrm{Ad}_{e^{[\boldsymbol{\mathcal{S}}_1]\theta_1}e^{[\boldsymbol{\mathcal{S}}_2]\theta_2}...e^{[\boldsymbol{\mathcal{S}}_{i-1}]\theta_{i-1}}}(\boldsymbol{\mathcal{S}}_i) \;\; i = 2, 3, ..., n \tag{2.27}$$

and the first column $\boldsymbol{J}_{s1} = \boldsymbol{\mathcal{S}}_1$. Similarly the body Jacobian of a n-joint manipulator is defined as:

$$\boldsymbol{\mathcal{V}}_b = \boldsymbol{J}_b(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \tag{2.28}$$

where the ith column of $\boldsymbol{J}_b(\boldsymbol{\theta})$ is:

$$\boldsymbol{J}_{si}(\boldsymbol{\theta}) = \mathrm{Ad}_{e^{-[\boldsymbol{\mathcal{B}}_n]\theta_n}e^{-[\boldsymbol{\mathcal{B}}_{n-1}]\theta_{n-1}}...e^{-[\boldsymbol{\mathcal{B}}_{i+1}]\theta_{i+1}}}(\boldsymbol{\mathcal{B}}_i) \;\; i = n-1, n-2, ..., 1 \tag{2.29}$$

and the last column $\boldsymbol{J}_{bn} = \boldsymbol{\mathcal{B}}_n$.

2.8.1. The Pseudoinverse of the Jacobian

It can be useful to find the inverse of the Jacobian matrix. For a six-joint open chain manipulator the Jacobian will be a 6×6 matrix and the inverse can be found normally, but this is not always the case. In the case where there are more than six joints, the Jacobian is called fat as it has more columns than rows. Accordingly it is called "tall" when there are less than six joints as the Jacobian then will have more rows than columns.

In both cases the Jacobian will then no longer be square and the regular inverse can no longer be calculated. It is then possible to use the Moore-Penrose inverse commonly called the pseudoinverse. For a matrix $\boldsymbol{A}$ of full rank, the pseudoinverse $\boldsymbol{A}^{\dagger}$ can be calculated by the following formulas:

$$\boldsymbol{A}^{\dagger} = \boldsymbol{A}^{T}(\boldsymbol{A}\boldsymbol{A}^{T})^{-1} \text{ if } \boldsymbol{A} \text{ is fat} \tag{2.30}$$

$$\boldsymbol{A}^{\dagger} = (\boldsymbol{A}^{T}\boldsymbol{A})^{-1}\boldsymbol{A}^{T} \text{ if } \boldsymbol{A} \text{ is tall} \tag{2.31}$$

2.9. Singularities

Singularities for open chain manipulators occur when it loses the ability to move its end-effector in one or more directions. Mathematically singularities occur when the Jacobin fails to be full rank. This happens in configurations where joint axes become linearly dependent. An example of a singularity can be seen in Figure 2.6 taken from [18]. In this case the wrist centre of the six-axis robot is placed directly above the shoulder joint making the four joint axes intersect in a common point, thus causing linear dependency.

2.10. Inverse Kinematics

The inverse kinematics problem can be regarded as the opposite of forward kinematics. Given an end-effector pose $\boldsymbol{T}(\boldsymbol{\theta})$ the task is to find a set of joint angles $\boldsymbol{\theta} = [\theta_1, \theta_2, ..., \theta_n]$ that will yield said pose.

Inverse kinematics can be calculated analytically and numerically. When solving the inverse kinematic problem analytically the geometry of the robot together with trigonometry is used to yield a solution. An exact analytical solution to the inverse kinematics problem can often be complicated and yield cumbersome calculations due to robot geometries not being ideal for analytical calculations. Simplifying the robot geometries for calculations will make the process simpler,

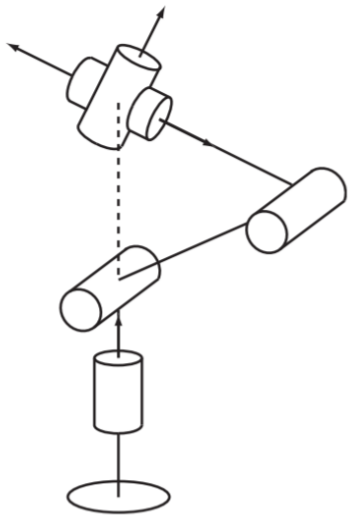


Figure 2.6.: A singular configuration for a six-axis open chain manipulator. Figure from [18].

but the solution will not be exact. In this case the joint angles can be found numerically using the analytical solution as an initial guess.

There are several numerical methods than can be used to solve the inverse kinematics problem. In [18], Newton's method is used as a basis for deriving the equations shown in this section.

To show how Newton's method is used to solve the inverse kinematics problem, consider a robot manipulator with an end-effector frame {b} and a fixed frame {s}. The desired end-effector configuration $\boldsymbol{T}_{sd}$ is given along with an initial guess for the values of $\boldsymbol{\theta}$ denoted $\boldsymbol{\theta}_0$. By applying the matrix logarithm the difference between the desired pose $\boldsymbol{T}_{sd}$ and the pose after i iterations $\boldsymbol{T}_{sb}(\theta_i)$ can be represented by a twist $\boldsymbol{\mathcal{V}}_b$ as:

$$[\boldsymbol{\mathcal{V}}_b] = \log(\boldsymbol{T}_{sb}^{-1}(\boldsymbol{\theta}_i)\boldsymbol{T}_{sd}) \tag{2.32}$$

A small deviation between the desired pose and the current pose of the end-effector will yield a small twist. In other words, the magnitude of the elements in $\boldsymbol{\mathcal{V}}_b$ can be used to measure the error in the estimation. Since numerical methods do not necessarily reach an exact solution, it is useful to set acceptable errors for the magnitude of the angular and linear velocity denoted ϵ_ω and ϵ_v respectively.

While $||\boldsymbol{\omega}_b|| > \epsilon_\omega$ or $||\boldsymbol{v}_b|| > \epsilon_v$, i is iterated and the angles $\boldsymbol{\theta}$ will get closer and

Result: $\boldsymbol{\theta}$
$i = 0, \boldsymbol{\theta} = \boldsymbol{\theta}_0$;
while $||\boldsymbol{\omega}_b|| > \epsilon_\omega \ or \ ||\boldsymbol{v}_b|| > \epsilon_v$ **do**
  $[\boldsymbol{\mathcal{V}}_b] = \log(\boldsymbol{T}_{sb}^{-1}(\boldsymbol{\theta}_i)\boldsymbol{T}_{sd})$;
  $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \boldsymbol{J}_b^\dagger(\boldsymbol{\theta}_i)\boldsymbol{\mathcal{V}}_b$;
  $i = i + 1$
end

Figure 2.7.: An algorithm for solving the inverse kinematics problem using Newton's method.

closer to the desired position, as described by the following formula:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \boldsymbol{J}_b^\dagger(\boldsymbol{\theta}_i)\boldsymbol{\mathcal{V}}_b \tag{2.33}$$

An algorithm for solving the inverse kinematics using (2.32) and (2.33) can be set up as seen in Figure 2.7.

Recall (2.28), and note that $\boldsymbol{J}_b^\dagger(\theta_i)\boldsymbol{\mathcal{V}}_b$ is the joint velocities needed to reach the twist $\boldsymbol{\mathcal{V}}_b$. Each increment in (2.33) takes the previous joint angles and adds the joint angle travelled in the time between each increment. This can be utilised for joint position robot control, by slowly changing the desired joint angles along a planned trajectory without letting the actual joint angles catch up. The previously desired angles can then be used as the initial guess for the numerical algorithm.

2.11. Inverse Velocity Kinematics

Inverse velocity kinematics deals with the problem of finding the necessary joint velocities $\dot{\boldsymbol{\theta}}$ to create a desired twist $\boldsymbol{\mathcal{V}}_d$ represented in an arbitrary frame. As previously mentioned the relation between an end-effector twist and the joint velocities are given by the Jacobian as shown in (2.26) and (2.28). Introducing the pseudoinverse yields the following:

$$\dot{\boldsymbol{\theta}} = \boldsymbol{J}^\dagger(\boldsymbol{\theta})\boldsymbol{\mathcal{V}}_d \tag{2.34}$$

The pseudoinverse used in (2.34) will prioritise all joint movement equally. To change this prioritisation the Jacobian can be altered with respect to different weighting functions. In [18] the Jacobian is altered so that the kinematic energy in the robot is minimised. Inverse velocity kinematics can be used in velocity control of robots.

Chapter 3.

Robotic welding

This chapter will focus on the hardware and software solutions used in robotic welding. Different welding methods will first be presented, before moving on to welding sensors. A brief overview of a selection of available welding robots will be given, before presenting different ways of programming welding robots.

3.1. Welding methods

Different welding methods are used in robotic welding depending on the welding task. This section will present a selection of welding methods used in robotic welding. The content of this section is based off the work done in [12] unless stated otherwise.

3.1.1. Arc Welding

Arc welding is the process of joining two metal pieces by utilising the heat from an arc generated between a welding gun and the workpiece. The welding gun consists of an electrode that is used to transfer current to the arc. Electrodes can be consumable or non-consumable. Consumable electrodes provide filler metal to the weld by melting itself, while non-consumable electrodes only transfers current and filler metal must supplied from an external source.

As filler metal and the metal in the work piece is melted during the welding process, it is highly susceptible to reactions with gasses in the air. To prevent this reaction from happening, a shielding gas can be deployed. This can either be inert gases like argon or helium, or active gasses like CO_2, depending on whether the gas is to react with the metal or not. Another alternative to protect the molten metal is the use of flux. This is a protective material added to the process that melts into slag which will cover the molten metal underneath to prevent

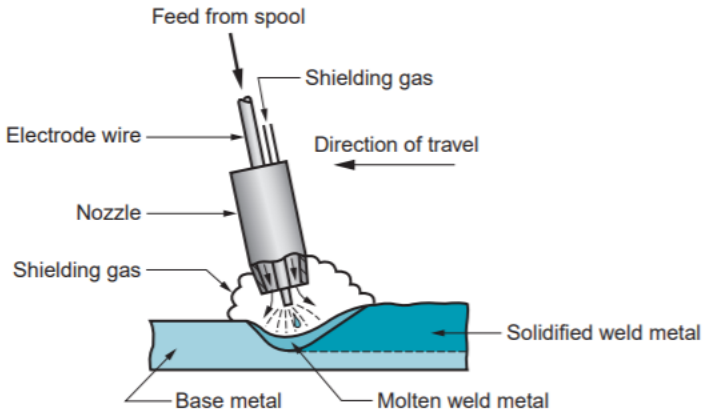


Figure 3.1.: A gas metal arc welding (GMAW) process. Figure from [12].

oxidisation. When the weld is cooled, the slag will harden and must be removed in post-processing of the workpiece.

A power source is required to generate the arc. Both DC and AC power is used. While AC is cheaper and generally easier to use, it is limited to welding ferrous metals. DC power sources are more expensive but can be used on all metals. This concludes the basic concept of arc welding, and different arc welding methods will now be presented.

MIG/MAG

Metal inert gas (MIG) welding and metal active gas (MAG) welding are both gas metal arc welding (GMAW) processes. MIG/MAG use a consumable electrode in the form of a wire that is continuously fed through the welding gun. This makes it an effective welding method for automation as it does not require change of electrode before the spool of wire has run out. An illustration showing a GMAW process has been taken from [12] and can be seen in Figure 3.1.

The shielding gas is provided through a tube in the welding gun, ad the difference between MIG and MAG lies in the composition of the gas. MIG uses inert gases that does not affect the welding process, while MAG uses active gases that do affect it. Because of the shielding gas, no flux is needed and GMAW therefore generates no slag.

CMT

Cold metal transfer (CMT) is a version of MIG welding, and is a relatively new welding method developed in 2004 by Fronius [30]. It differs from MIG welding by how the current is controlled in coordination with the motion of the electrode wire. To explain the process Figure 3.2 has been taken from [30]. The process

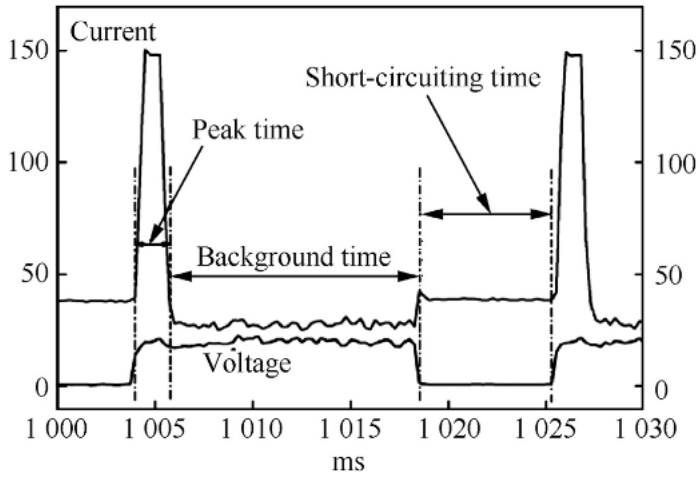


Figure 3.2.: The current and voltage during one cycle of a cold metal transfer process. Figure from [30].

description is also based on the work done in [30].

CMT welding can be regarded as a cycle process where a cycle is defined from one current peak to the next. In Figure 3.2 one cycle lasts approximately 20ms. At the start of each cycle the current spikes to ignite the arc before it is lowered again while the tip of the metal wire melts and forms a droplet. As the droplet makes contact with the molten metal in the welding pool, the welding circuit is shorted and the voltage goes to zero. This will signal the welding gun to retract the wire fast, which removes the molten droplet from the wire and adds it to the welding pool. After adding the droplet to the pool the cycle is completed and the arc is reignited at the start of the next cycle.

In comparison to traditional MIG welding, CMT transfers less heat to the work piece resulting in less deformations. CMT welding also benefits from less splatter due to the low currents compared to traditional MIG welding.

TIG

Tungsten inert gas (TIG) welding is another arc welding method similar to GMAW. The difference being that TIG uses a non consumable tungsten electrode, so filler material is not necessarily added. If filler metal is added it is done by an external metal rod that is melted by the heat of the arc. A TIG process can be seen in Figure 3.3 taken from [12].

Inert shielding gas is used in TIG welding. Just like with GMAW this eliminates the need for flux which again prevents the formation of slag, so little to no post-

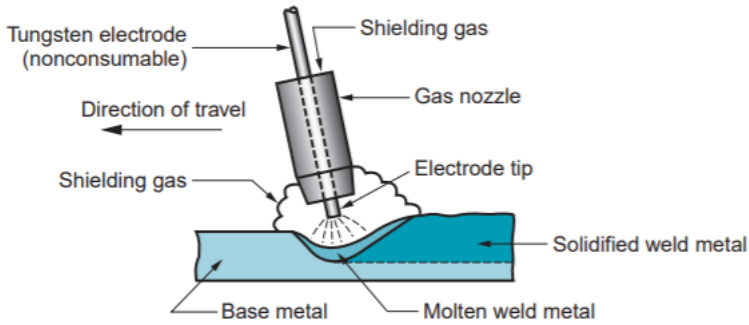


Figure 3.3.: Tungsten inert gas (TIG) welding. Figure from [12].

processing is required. TIG is generally slower and more costly than GMAW, and also harder to automate due to the external filler metal. On the other side TIG yields a higher quality weld, is better for welding thin plates where no filler metal is required and does not produce any spatter as the filler metal is not transferred over the arc.

3.1.2. Resistance Welding

Resistance welding is a group of welding processes that use a mixture of heat and pressure. Two pieces of metal are placed on top of each other and pressed together by two electrodes before a current is applied which will flow through the work pieces. The electrical resistance that occurs between the two workpieces will generate enough heat to melt or soften the metal, and the welding occurs. This method uses high currents (5000-20 000A), but voltages are kept relatively low (below 10V). Resistance occurs several places in the process:

- In the electrodes
- In the workpieces
- Between the work pieces and the electrodes
- Between the work pieces

As the goal of the process is to weld together the workpieces, it is preferred that most of the resistance occurs between the two workpieces. This is done by reducing resistance elsewhere. Highly conductive metals are used in the electrodes, e.g. copper, to reduce resistance. To reduce the resistance between the electrodes and the workpieces, clean surfaces and good surface contact is important.

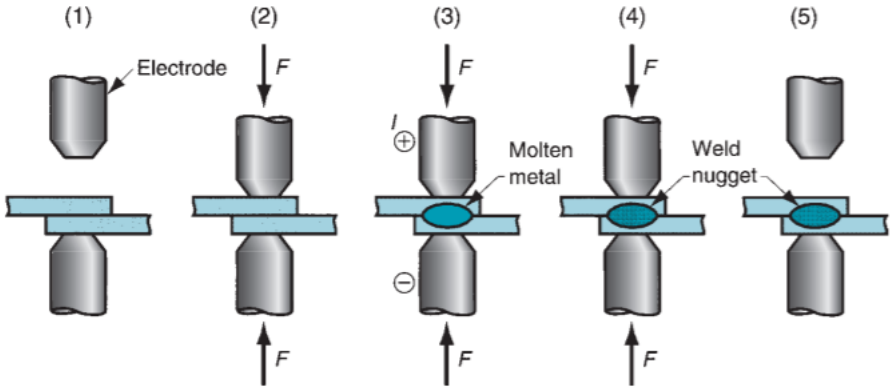


Figure 3.4.: The resistance spot welding (RSW) process. A force F is applied (2-4), before a current is applied (3). Figure from [12].

RSW

Resistance spot welding (RSW) is the most common type of resistance welding, and is widely used for tasks where metal sheets of 3mm or thinner are to be welded. The welding apparatus differ depending on how the welding method is implemented. For robotic automation, a handheld device is used. These are called portable spot-welding guns. They consist of two opposing electrodes that can be pinched together to clamp the workpieces and create the required pressure. Current is then applied through the electrodes, and the welding occurs. The welding process can be seen in Figure 3.4 taken from [12].

RSW has the advantage of being an effective welding process yielding high production rates without the need for filler metal. It is also a reliable and repeatable process that is easily automated in comparison to e.g. arc welding. The downside to RSW is expensive equipment and the fact that it can only weld overlapping workpieces.

3.1.3. Laser Welding

As the name suggests, laser welding or laser-beam welding (LSB) is a welding method where a laser is used to generate heat. Shielding gasses may be used to protect the weld when welding reactive metals, and usually no filler metal is used. Due to the concentrated heat generated by a laser, LSB generates narrow and deep weld seams. This makes the process ideal for welding smaller parts.

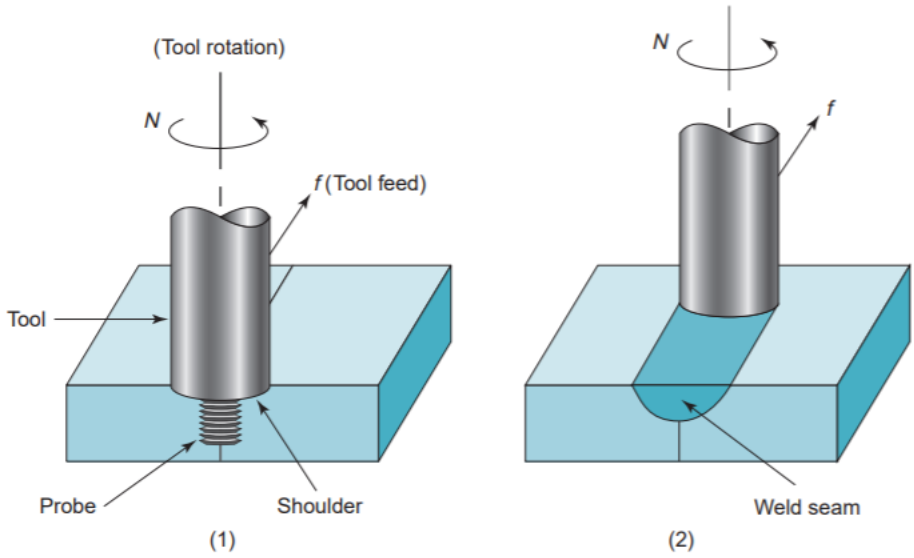


Figure 3.5.: The friction stir welding (FSW) process. The tool is first inserted into the workpiece (1) before it is fed in between the pieces to be welded (2). Figure from [12].

3.1.4. Friction Stir Welding

Friction stir welding (FSW) is a welding method in the category of solid-state welding, which are methods that rely on pressure and heat to soften but not melt the metal. No filler metals are required in solid-state welding. The FSW process can be seen in Figure 3.5 which is taken from [12].

The FSW tool consists of a a shoulder and a probe. The probe is inserted in between two pieces of metal that are to be welded, while the shoulder presses on top of the pieces. The tool is then rotated and fed along the welding groove. The friction between the shoulder and the surface of the workpieces generates enough heat to soften the metal. Heat is also generated from the rotation of the probe. Along the probe there are threads like the ones found in screws. The threads stir the softened metal and "mixes" the two workpieces together resulting in a welding seam. FSW creates welding seams with good mechanical properties, little distortion in the workpiece and does not depend on shielding gases or flux like arc welding does. The downside to FSW is the need for heavy clamping of the workpieces to keep them in place during the process, and the exit hole that is produced when the probe is extracted from the workpiece.

3.2. Welding Sensors

During the welding process the generated heat will cause the metal in the work-piece to thermally expand. In a mass production each individual work-piece also may not have the exact same orientation when clamped down to be welded. Because of these variables it is beneficial to introduce sensors to robotic welding operations. The main task for sensors in robotic welding is seam tracking. This is the task of keeping track of the welding groove to make corrections to the robot trajectory if the groove deviates from the originally programmed end-effector trajectory. This section will explain two different sensor technologies in robotic welding: through-arc sensing and optical sensing.

3.2.1. Through-Arc Sensing

Through-arc sensing is a seam tracking method used in arch welding where the welding parameters are utilised to keep track of the welding groove. While welding it is possible to measure the current and voltage. From [23] the arc characteristic model is is obtained:

$$U = \beta_1 I + \beta_2 + \beta_3 / I + \beta_4 l \tag{3.1}$$

Where U is the arc voltage, I is the arc current, l is the distance from the welding gun to the work piece and the constants $\beta_1 - \beta_4$ are determined based on characteristics of the welding equipment. It is common for welding power supplies to keep a constant voltage, which leaves (3.1) with only the welding current I and the distance l as variables. By measuring the arc current it is then possible to estimate the distance l. These two variables are approximately negative proportionate.

While calculating the length l different algorithms can be implemented to follow a welding trajectory. A common way of implementing through-arc sensing with robots is to utilise an oscillatory motion of the welding gun while following the planned trajectory. This will detect any discrepancies in the weld groove and allow the robot to correct its movement. An example of this method being used can be seen in Figure 3.6 from [23] where the parts that are to be welded are not perfectly aligned. The robot is able to detect the errors and correct its path using oscillatory movements in combination with (3.1).

The algorithms for seam tracking using though-arc sensing are dependent on a robust robot controller. If the end-effector deviates too much from the welding trajectory, the sensor might not be able to recover from the error. Also keep in mind that the sensor only works while the welding process is active, so after losing

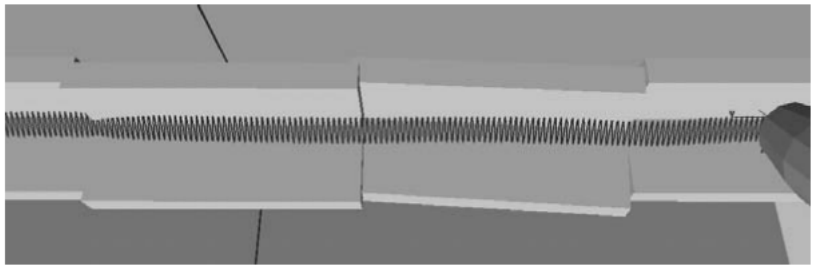


Figure 3.6.: Example of a robotic welding process using through-arc sensing to correct the welding path. Figure from [23].

track of the trajectory it can be hard to recover.

In practice through-arc sensing is often combined with some kind of search algorithm or other sensing method to find the start and orientation of the welding groove. From [4] it is seen that a way of doing this is by touch sensing. This is a method there the tip of the welding gun can be used to detect the orientation of a part by applying a small voltage to the electrode and make contact with the part. When contact is obtained the circuit is shorted and the point in 3D can be saved to the robot program. By measuring enough points the orientation of a part can be achieved and the welding trajectory can be recalculated.

3.2.2. Optical Sensing

For optical seam tracking, cameras and image processing are used to calculate and correct the robot trajectory before or during the welding process. When using optical sensors, the camera is usually mounted on the end-effector of the robot. Before the camera can be put to use, a calibration is necessary to obtain the cameras extrinsic and intrinsic parameters. A hand-eye calibration is also necessary to find the relation between the camera frame and the end-effector frame of the robot. Calibration can be done by analysing a series of pictures of chequerboards with known dimensions.

In [28] two optical sensors are described in most detail, these are vision sensors and laser assisted vision sensors. This section will do a brief summary of the description of vision senors, before a more thorough description of laser assisted vision sensors will be given.

Vision sensors use computer vision to detect the weld groove. After camera calibration, the camera is able to capture images of the groove. These images are then processed in several steps where noise is removed and the features of the welding groove are extracted. As the relation between the camera frame and the

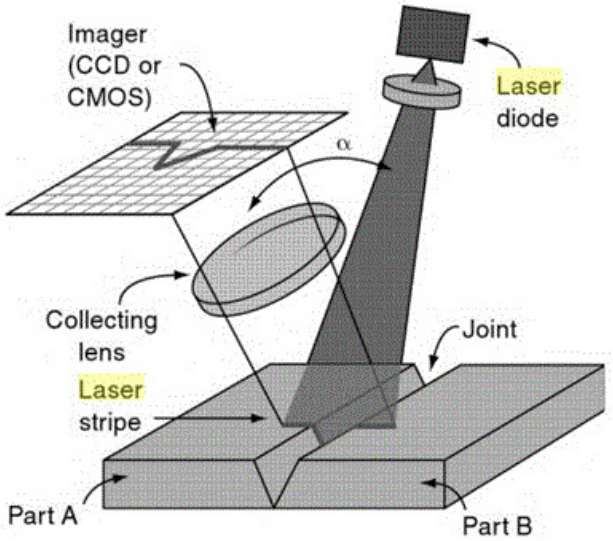


Figure 3.7.: The concept behind laser vision sensors. A projected laser image is used by the camera to find the features of the welding joint. Figure from [32].

robot end-effector frame is known from calibration, it is possible to adjust the welding trajectory based on the processed image. The orientation of a workpiece can also be detected in the same manner.

The problem with vision sensors are the complexity of the algorithms for extracting 3D groove-parameters from the acquired 2D images [28]. This can be solved by adding a laser to the setup, this is the case for laser vision sensors.

To explain laser vision sensors consider Figure 3.7 from [32] where the basic setup for the sensor is shown. A laser diode emits a known shape onto the welding groove, labelled "joint" in the figure. This is usually in the form of a single or multiple lines, but can also be other shapes. According to [23] the use of a circle instead of lines can be useful for detecting corners in one image. The rest of this description will be based on the single line configuration seen in Figure 3.7.

The projected laser line is captured by the camera, labelled "imager" in the figure. The image is then processed to remove noise and extract the features of the laser. Since the angle α is known it is possible to calculate the dimensions of the welding groove by triangulation. The use of triangulation replaces the complex algorithms used in regular vision sensors which makes laser vision sensors faster. This is important when seam tracking while welding.

The laser vision sensor is a reliable sensor which can extract different useful weld groove features that can be taken into consideration when planning the end-

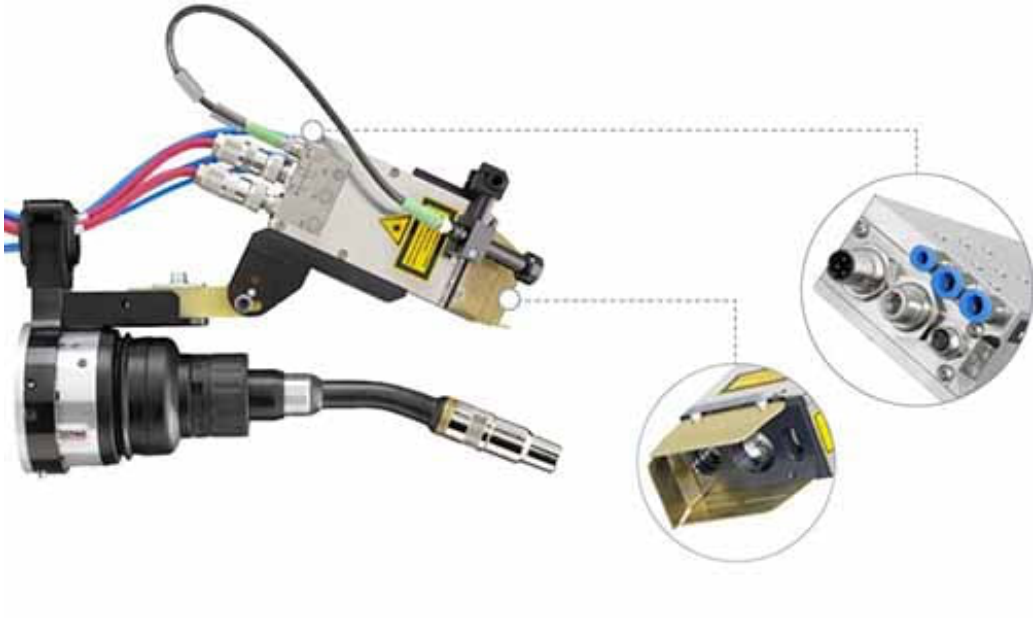


Figure 3.8.: The TH6x optical seam tracking sensor mounted on a welding gun. Image from [6]

effector trajectory, wire feed and welding speed. The problem with the sensor is the required space it occupies on the end-effector as can be seen in Figure 3.8 from [6] which can limit the mobility of the robot. The optical sensors are more expensive than using though-arc sensing. Optical sensors are therefore usually avoided unless they are necessary for the task [23].

3.3. Welding Robots

This section will present a selection of welding robots from some of the biggest robot manufacturers. This will give insight to the hardware that is available for the industry today. The technical details of the selected welding robots can be seen in Table 3.1, these are gathered from [11][37][1][14]. Pictures of the different robots can be seen in Figure 3.9. It is seen that welding robots are made in different sizes depending on their intended use. Different operations require different load capacities as the weight of welding gear can differ. Common for all the welding robots mentioned in this report are the number of axis, which is six. The reason for this can be seen by applying Grübler's formula (2.1) to the robots. As they only have rotational joints the number of DOF is equal to the number of joints.

In Section 2.1 it is explained that a rigid body in 3D has six DOF. By having

Producer	Model	Nr. of axis	Loading capacity[kg]	Reach[mm]
FANUC	ARC Mate 100iD	6	12	1441
Yaskawa	MOTOMAN AR900	6	7	927
ABB	IRB 1300	6	7-11	900-1400
KUKA	KR QUANTEC prime	6	90-240	2496-3701

Table 3.1.: Technical details for a selection of welding robots. Data from [11][37][1][14].

six-axis robots the end-effector is able to move freely in 3D, of course limited by the features of the robot. Enabling end-effector movement with six DOF makes industrial robots versatile and able to complete a variety of different tasks. Some industrial robots are made with more than six axis, like the YASKAWA SIA Series which have seven. This will cause redundancy as discussed in Section 2.2.

Figure 3.9.: The robots listed in Table 3.1, note that the relative scale of the images are not correct. From the left: ARC Mate 100iD, MOTOMAN AR900, KR 240 R2500 prime, IRB 1300. Images from [11][37][1][14]

.

3.4. Robot Programming

Programming methods for industrial robots can be categorised as either online robot programming or offline robot programming. While many different ways of robot programming are suggested in literature [5], these are often experimental

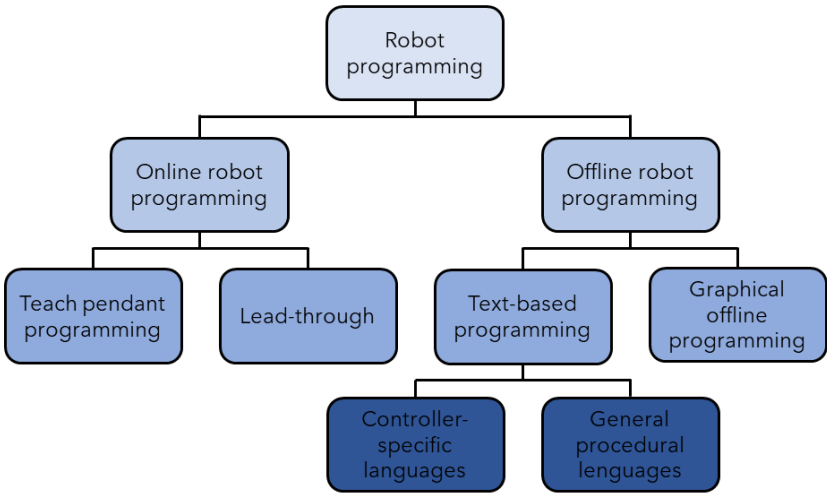


Figure 3.10.: The different robot programming methods presented in this report.

and not necessarily ready for the industry. This section will present some of the commonly used methods for programming welding robots and explain the advantages and drawbacks with each method.

3.4.1. Online Robot Programming

When doing online robot programming the physical robot is incorporated in the programming process. While this gives the programmer a physical understanding of the program being made, it also requires the robot to be taken out of production. This is not always an option if the robot is part of e.g. a large serial production. Online programming is generally easier than offline programming and usually requires little programming skills. Two main methods are used in online programming: teach pendant programming and lead-through [22].

Teach Pendant Programming
The teach pendant is a handheld device that is usually attached to the robot controller with a cable. A teach pendant form KUKA can be seen in Figure 3.11. It allows the user to directly communicate with the robot and control it freely. For safety reasons the robot joint velocities are limited when operated manually by the teach pendant.

Teach pendant programming is done by driving the robot manually to points of interest and save the joint coordinates. The finished program is made by combining movements between the saved points in sequence.

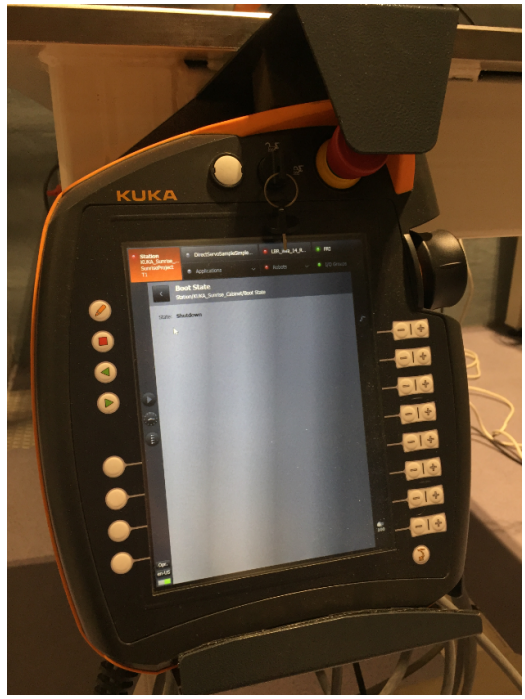


Figure 3.11.: Teach pendant from KUKA.

Lead-Through
This is a manual way of programming a robot where the operator "teach" the robot how it is supposed to move. This is done by manually grabbing and dragging the end-effector of the robot through the desired task and record the movement.

This is an intuitive way of programming robots and is often used for welding or spray paint tasks. The drawback using lead-through programming is that it is inaccurate compared to other programming methods, and it might require the robot to be equipped with force sensors.

3.4.2. Offline Robot Programming

Offline robot programming is a type of programming were the physical robot is not necessarily included in the programming process. The robot program is made separately and can be tested in a simulator before uploading it to the robot. This may require separate simulation software which can be expensive, but the advantage of offline programming is that the robot can operate as normally while the program is being developed. According to [22], offline programming can be divided int two different categories.

Text-Based Programming
This is a traditional method of robot programming where the robot program is written with text, before uploading it to the robot. Text-based programming can be divided into three different categories based on the type of language used: controller-specific languages, generic procedural languages and behaviour-based languages [5]. For the industrial robots discussed in this report, the two former methods are the most relevant and will be discussed further.

It is common for robot manufacturers do develop their own programming language suited for their robot models. By using this language for programming there will be access to built in functions made by the manufacturer, and the programming task can be simplified. The drawback of this method is the lack of a universal programming standard for all robot manufacturers to follow. This makes each controller-specific language different, and having robots from multiple manufacturers will require knowledge with different programming languages.

Using a generic procedural language yields opportunities for making more universal robot programs that can be implemented with different robot brands. It is the preferred robot programming method for researchers as it requires flexibility to meet the needs of the research being conducted [5]. Before implementing the program on a robot it has to be translated into the controller-specific language. General procedural languages will often require more programming knowledge than the controller-specific languages.

The general drawback of text based robot programming is that it requires the most debugging [22].

Graphical Offline Programming
Graphical offline programming can be regarded as a combination of teach-pendant programming and offline programming. A CAD-model of the robot is used to simulate motions. Robot movements can be simulated and recorded to generate a program for the physical robot. This offline programming method have the advantage of giving the user a visual feedback on the program that is created, making it a possibly more intuitive offline programming method than text-based programming.

Chapter 4.

Constraint-Based Robot Programming

This chapter will first present constraint-based programming in general, and give a brief introduction to how this can be used in robotics. This will be followed by a thorough summary of three different frameworks for constraint-based robot programming, before a brief summary of other frameworks will be given. Lastly the work done with constraint-based robot programming in robotic welding will be presented.

4.1. Constraint-Based Programming

Constraint-based programming is a programming method where a problem is defined by a set of constraints, and the solution to the problem is a set of variables that is to be found. The constraints limit the possible values of the variables, called the variable domains. By feeding the variables and constraints into a constraint solver, variable values that satisfy the given constraints can be found. An example of a simple constraint-based problem can be found in [26] and is given as:

$$x, y, z \in \{0, 1\} \tag{4.1}$$

$$x^2 + y^2 = z^2 \tag{4.2}$$

In this case, the variables are x, y, z. They each have their own domain D_x, D_y, D_z respectively, which are all given in (4.1) as $\{0, 1\}$. For this problem there is a single constraint which can be seen in (4.2). This is a simple example with only

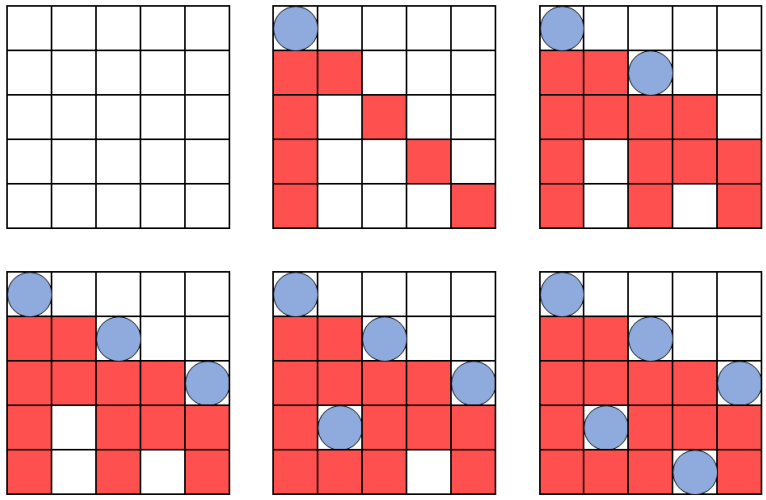


Figure 4.1.: Constraint propagation used to solve a problem where tokens are to be placed on each row, adapted from [26].

three solutions, but constraint-based problems usually have more. To reduce the number of solutions, constraint propagation is an important concept [27].

Constraint propagation is the process of reducing the domain of variables by using the constraints present in the problem. A simple example of constraint propagation can be seen in [26] and will be presented here. In this example n tokens are to be placed on separate rows in a $n \times n$ grid. None of the tokens are allowed to be in the same column or on the same diagonal. By placing a token on the first row at random, it is possible to identify invalid placements on the other rows. This will decrease the number of valid tiles for the rest of the tokens. In this example, constraint propagation alone is enough to yield a unique solution to the problem. This can be seen in Figure 4.1 where the problem is solved for a 5×5 grid.

If constraint propagation does not solve the problem by itself, one of two different algorithms are mainly used [27]. These are backtracking search and local search, and will now be explained.

To explain backtracking search, the example with the $n \times n$ grid is considered again. Placing the first token randomly, different solutions can be attempted by placing the second token. If a valid placement for the second token is found, the algorithm moves on to the third and so on. Each time a non valid solution is found, the algorithm backtracks to the last valid placed token and tries a new

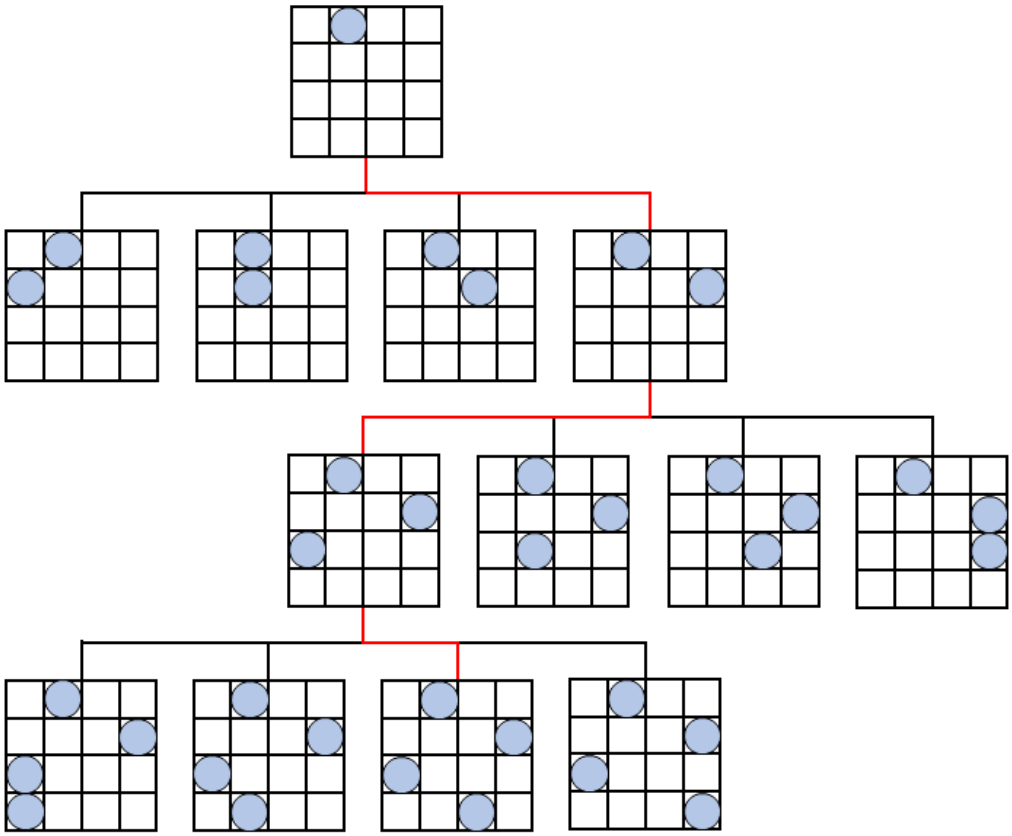


Figure 4.2.: A visualisation of the backtracking algorithm. The path to a valid solution is shown by the red line, adapted from [26].

solution. This will result in a tree like structure like the one seen in figure 4.2 where one branch of the "tree" is explored. A red line marks the path a valid solution. The advantage of using a backtracking search is that it will always find the optimal solution, but it could require a lot of computation. To reduce the complexity of the computation, constraint propagation can be utilised.

Local search is another method of finding a solution. This method is not necessarily guaranteed to find the optimal solution, but is less computation heavy than backtracking search. This method starts with all variables chosen at random. In turn the algorithm changes one variable at the time. If changing a variable yields a solution that is closer to fulfilling the constraints the change is kept, if not it is reverted. This process is repeated until a satisfying solution is found. Applying this algorithm the example used previously, this would be equal to first placing all tokens at random on the grid. By moving one token at the time, a solution

will be found.

4.2. Constraint-Based Robot Programming, a Selection of Frameworks

When applying constraint-based programming to robotics, constraints are set either from robot characteristics, the environment or the task to be executed. These constraints are then sent into a constraint solver, and a robot trajectory can be calculated. This section will present three different frameworks that has been developed for constraint-based robot programming, and some of the theory behind them.

4.2.1. iTaSC

Instantaneous Task Specification using Constraints (iTaSC) is a framework for robot control developed at K.U Leuven [21]. This section is a summary of [8] where the concepts behind iTaSC is explained. The main focus lies with specifying the task that is to be completed by the robot, but a control scheme is also suggested. The framework is further expanded in [9] which will be summarised at the end of this section.

The goal of iTaSC is to enable robot programming of more complex tasks. While programming of simple robot tasks in well known environments are straightforward and already used in the industry, there is little programming support for more complex tasks. This requires the robot programmer to be proficient in multiple disciplines like e.g. spatial kinematics, estimation and sensor systems. iTaSC aims to make programming easier by providing a framework for defining the task at hand. Based on this task specification a robot trajectory can be found. The form of task specification used in iTaSC is called constraint-based task programming, which is presented in [29]. The iTaSC framework also aims to represent geometric uncertainties, from the model or robot environment, in a systematic way.

Control Scheme and Equations
The general control scheme for iTaSC can be seen in Figure 4.3. The figure shows the plant P representing the robot itself, the controller C and an update and estimation block M+E where estimated states are calculated and sent to the controller.

Several variables are seen in the control scheme. $\boldsymbol{u}$ is the input to the system and is either desired joint velocities, acceleration or torques depending on how the robot is controlled. $\boldsymbol{\chi}_u$ is geometric disturbance in the system, e.g. deviation in the

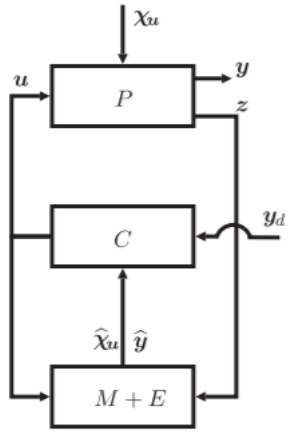


Figure 4.3.: The control scheme used in iTaSC showing the plant P, the controller C and the update and estimation block M + E. Figure from [8].

position of a work piece. $\boldsymbol{y}$ is the output of the system and can be joint velocities, joint positions, forces, torques, distances etc. $\boldsymbol{z}$ represents the measurements from the robot that are sent to the estimator. $\boldsymbol{y}_d$ represents the desired values of $\boldsymbol{y}$ and therefore also represents the constraints on the system. In addition to the variables seen in figure 4.3, two more variables are defined. Joint positions are represented by the variable $\boldsymbol{q}$. There are also feature coordinates represented by $\boldsymbol{\chi}_f$, which are used to describe the task that is to be completed.

Equations (4.3) - (4.8) as well as their explanations are taken from [8]. These are the equations used in the controller. For this explanation it is assumed that the robot is controlled using joint velocities $\dot{\boldsymbol{q}}$. The robot joint positions $\boldsymbol{q}$ and joint velocities $\dot{\boldsymbol{q}}$ are related to the input $\boldsymbol{u}$ by the robot system equation:

$$\frac{d}{dt}\begin{pmatrix}\boldsymbol{q}\\ \dot{\boldsymbol{q}}\end{pmatrix} = s(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \tag{4.3}$$

The output $\boldsymbol{y}$ is calculated by the function f in this manner:

$$f(\boldsymbol{q}, \boldsymbol{\chi}_f) = \boldsymbol{y} \tag{4.4}$$

The measurements are calculated similarly by the function h:

$$h(\boldsymbol{q}, \boldsymbol{\chi}_f) = \boldsymbol{z} \tag{4.5}$$

In [8] there is a distinction between natural and artificial constraint. The natural constraints are imposed on the system by robot limitations or obstacles in the robot environment, and can be expressed as:

$$g(\boldsymbol{q}, \boldsymbol{\chi}_f) = \mathbf{0} \tag{4.6}$$

Artificial constraints on the other hand are imposed on the robot by the user by setting desired output values as such:

$$\boldsymbol{y} = \boldsymbol{y}_d \tag{4.7}$$

The relation between $\boldsymbol{q}$ and $\boldsymbol{\chi}_f$ will be affected by the uncertainty coordinates $\boldsymbol{\chi}_u$. The equation describing this relation is:

$$l(\boldsymbol{q}, \boldsymbol{\chi}_f, \boldsymbol{\chi}_u) = 0 \tag{4.8}$$

Frames and Coordinates
iTaSC relies heavily on defining different frames and the relation between them. There are two different kind of frames. The object frames are attached to rigid objects either on the robot itself or in the robot environment. A feature frame is attached to a feature on the object such as an edge, a surface or an abstract feature such as the axis of a hollow cylinder. Two object frames ($o1$ and $o2$) and two feature frames ($f1$ and $f2$) represent one constraint. This constraint will specify the relative movement of two objects, or the forces acting between them. There are four rules in total for choosing object and feature frames, these are stated in [8] as:

- $o1$ and $o2$ are rigidly attached to the corresponding object.
- $f1$ and $f2$ are linked to, but not necessarily rigidly attached to $o1$ and $o2$, respectively.
- The origin and orientation of the frames are chosen as much as possible such that the submotions correspond to motions along or about the axes of one of the two frames involved in the submotion.
- Every system output and every measurement is modelled using a feature relationship. However, a single feature might be used to model multiple system outputs and/or measurements.

The movement between the four frames that define a constraint defines a kinematic chain from one object frame to the other. The movements are gathered in the matrix $\boldsymbol{\chi}_f$ in the following manner:

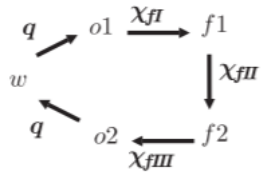


Figure 4.4.: The kinematic chain constraining one object from to the other. Figure from [8].

$$\boldsymbol{\chi}_f = \begin{pmatrix} \boldsymbol{\chi}_{fI}^T & \boldsymbol{\chi}_{fII}^T & \boldsymbol{\chi}_{fIII}^T \end{pmatrix}^T \tag{4.9}$$

where [8] describes the components as:

- $\boldsymbol{\chi}_{fI}^T$ represents the relative motion of $f1$ with respect to $o1$.
- $\boldsymbol{\chi}_{fII}^T$ represents the relative motion of $f2$ with respect to $f1$
- $\boldsymbol{\chi}_{fIII}^T$ represents the relative motion of $o2$ with respect to $f2$

The kinematic chain consisting of the three different motions will constrain one object frame with respect to another so that they can be uniquely determined from each other. Each motion consists of either translation along or rotation about one coordinate axis. Together the three movements will impose six constraints on the object frames. This is sufficient as a rigid body in space has six DOF as seen in Section 2.1. Figure 4.4 from [8] shows how the frames are connected by the different motions.

Wile Figure 4.4 assumes no uncertainties in the system, this is an approximation and is not completely accurate. The concept of geometric uncertainty is represented in a similar manner to the feature coordinates, namely by the uncertainty coordinates. These are movements between the frame position in the model, and the frame position after applying uncertainty. The uncertainty coordinates are gathered in the matrix $\boldsymbol{\chi}_u$ and are written as:

$$\chi_u = \begin{pmatrix} \boldsymbol{\chi}_{uI}^T & \boldsymbol{\chi}_{uII}^T & \boldsymbol{\chi}_{uIII}^T & \boldsymbol{\chi}_{uIIII}^T \end{pmatrix}^T \tag{4.10}$$

Analogous to the feature coordinates, the uncertainty coordinates represents the following from [8]:

- $\boldsymbol{\chi}_{u1}$ represents the pose uncertainty of $o1$.
- $\boldsymbol{\chi}_{uII}$ represents the pose uncertainty of $f1$ with respect to $o1$.

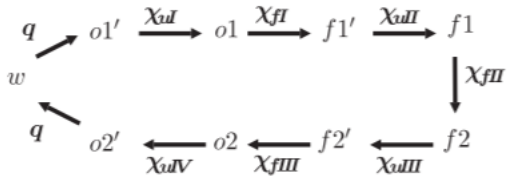


Figure 4.5.: The kinematic chain between the two object frames with uncertainty coordinates. Figure from [8].

- $\boldsymbol{\chi}_{uIII}$ represents the pose uncertainty of $f2$ with respect to $o2$.
- $\boldsymbol{\chi}_{uIV}$ represents the pose uncertainty of $o2$.

While also taking the uncertainty coordinates into account, the kinematic chain in Figure 4.4 is extended and the kinematic chain in Figure 4.5 is obtained.

After defining the frames by the four defined rules, a task is constrained by χ_f and χ_u . The coordinates along with the constraints are input to the control scheme, and the trajectory is calculated.

Extended Version
iTaSC was extended in [9]. In it's original form, the framework only dealt with equality constraints. These are constraints where the goal of a variable is to be equal to a specific value. In the extended form derived i [9], iTaSC could also handle inequality constraints. These are constraints where the goal for a variable is to be lower or higher than a limit.

Instantaneous task specification which is optimising a variable at the end of the movement, was previously the only option available in iTaSC. In [9] suggestions were made to implement non-instantaneous task specification where a variable could be optimised over time, though it was concluded that this would need further research.

4.2.2. eTaSL/eTC

The framework summarised in this section is described in [2]. The article presents the expression-graph Task Specification Language (eTaSL), which is a language made for expressing robot tasks. In addition to eTaSL [2] presents the expressiongraph-based Task Controller (eTC), which is a robot controller made to control a robot based on the task described in eTaSL or other task descriptions. This section will first describe the architecture of eTC before explaining important concepts of eTaSL.

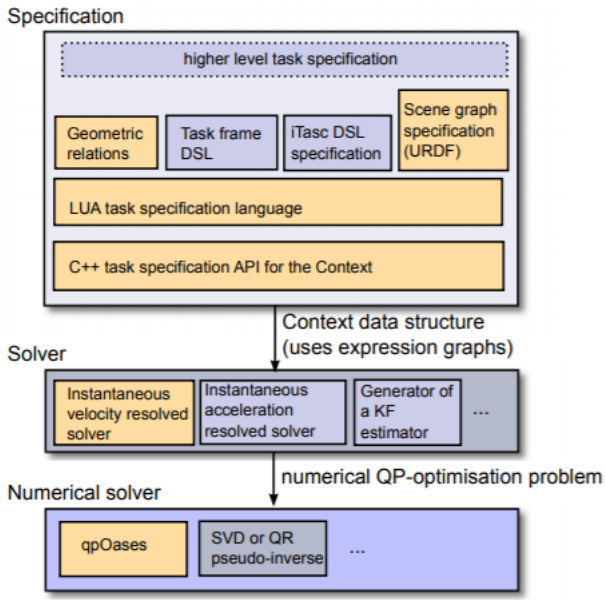


Figure 4.6.: The layers used in the eTC architecture, highlighted boxes are the implemented solutions. Figure from [2].

Robot Controller, eTC

The architecture of the robot controller focuses on the 5C's found in [24], and separates computation, coordination, configuration, composition and communication. Figure 4.6 from [2] shows that the controller has been separated into three layers, each with their own separate task.

The first layer is the specification layer where the context is built. This is a complete description of the task. Robot geometry can be loaded into the specification layer through a unified robot description format (URDF) file. This file will define the robot links, their geometric relations and robot constraints like joint angles and velocities. The task can be specified with a C++ task specification API, other task specification methods like iTaSC or more commonly with eTaSL. After specifying a task, the context is sent as input to the second layer.

The second layer is the solver layer, where a context is converted to a numerical optimisation problem. Referring to Figure 4.6 it can be seen that the current implementation uses a solver to control the joint velocities of the robot. eTC can also be used for acceleration control or torque control, but for this presentation velocity control is assumed. The optimisation problem has a goal of minimising the variable $\boldsymbol{x}$ in the following formula:

$$\boldsymbol{x}^T \boldsymbol{H} \boldsymbol{x} \tag{4.11}$$

with respect to the upper and lower limits $\boldsymbol{U}$ and $\boldsymbol{L}$ such that:

$$\boldsymbol{L}_A \leq \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{U}_A \tag{4.12}$$

$$\boldsymbol{L} \leq \boldsymbol{x} \leq \boldsymbol{U} \tag{4.13}$$

Where $\boldsymbol{A}$ is the task Jacobian and $\boldsymbol{L}_A$ and $\boldsymbol{B}_A$ are task limits. The variable $\boldsymbol{x}$ is written as:

$$\boldsymbol{x} = \begin{bmatrix} \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{\chi}}_{\boldsymbol{f}} \\ \boldsymbol{\epsilon} \end{bmatrix} \tag{4.14}$$

Where $\dot{\boldsymbol{q}}$ is the joint velocities, $\dot{\boldsymbol{\chi}}_f$ is the feature variable velocities and $\boldsymbol{\epsilon}$ is a slack variable to enable lower priority constraints to act on the system. The matrix $\boldsymbol{H}$ seen in (4.11) is written as:

$$\boldsymbol{H} = \begin{bmatrix} \mu \boldsymbol{W}_r & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \mu \boldsymbol{W}_f & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \mu \boldsymbol{I} + \boldsymbol{W}_s \end{bmatrix} \tag{4.15}$$

Where $\boldsymbol{W}_r$ and $\boldsymbol{W}_f$ are weights for the robot joint space and feature space respectively. $\boldsymbol{W}_s$ is a collection of the weight variable in each constraint, which will be presented later, while μ is a numerical value that can be used to tune the system. $\boldsymbol{W}_r$ and $\boldsymbol{W}_f$ will mostly affect the system when task redundancy occurs. In this case they will affect how the robot moves in the null-space of the task, i.e. how the rest of the robot is moved while holding the end-effector in the desired configuration.

After the optimisation problem is defined it is sent into the third and last layer where it is solved by a numerical solver, in this case the qpOASES solver. For each time cycle the joint positions are taken in as an input to the solver layer where the optimisation problem is generated before being sent to the numerical solver. The computed joint velocities are then sent to the robot. This process is repeated each time cycle.

Specification Language, eTaSL

eTaSL is a Lua based language using abstractions to communicate with the un-

```
Variable {
  context    = ctx,
  name       = 'along_path',
  vartype    = 'feature',
  weight     = 1.0
}
```

Figure 4.7.: Example of a variable in eTaSL. Figure from [2].

```
Constraint{
 context  =  ctx,
 name     = 'point_to_point_distance',
 expr     = norm(origin(arm)-origin(trajectory)),
 target   = 0.0,
 K        = 4,
 weight   = 1.0,
 priority = 2
}
```

Figure 4.8.: Example of a constraint in eTaSL. Figure from [2].

derlying C++ API generating the context in the specification layer. The context consists of different elements that will now be explained, starting with variables.

An example of a variable can be seen in Figure 4.7 taken from [2]. Variables are set by the user and is required to be connected to a context and have a name. They also need a weight which will affect the way they are handled by eTC. The `vartype` argument decides what variable type the variable is. Robot joint variables determines the joint angles in the robot, the time variable keeps track of time and a feature variable can be used more freely to express movements.

Another type of variable is the expression variable. This is a variable containing an expression graph, which is a function often representing geometric relations between rigid bodies. Expression variables are used as an argument when defining constraints and monitors, this will be presented next.

An example of a constraint can be seen in Figure 4.8 taken from [2]. Like variables it has to be connected to a context, have a name and a weight. In addition to this it uses an expression variable to define a function. The goal of the expression variable is to reach the value of `target`, in this case 0.0. How the expression variable will reach the target is determined by the value of `K`, which is used in the controller. Lastly the constraint has a priority. This allows the user to define a hierarchy were constraints with high priority are more important than the ones with a lower priority. For multiple constraints with equal priority, their importance is measured by the `weight` argument.

Monitors can detect when a certain condition is met and notify the system of

```
Monitor {
   context = ctx,
   name    = "goal_reached"
   expr    = norm( origin(arm)-origin(goal) ),
   lower   = 1E-4,
   actionname     = "event",
   argument       = "e_goal_reached"
}
```

Figure 4.9.: Example of a monitor in eTaSL. Figure from [2].

this event. An example of a monitor can be seen in Figure 4.9 taken from [2]. It is seen that the monitor observes an expression variable defined by `expr`. As this expression exceeds some limit `lower` or `upper`, an action is triggered and the argument is sent as an output. The name of the action and the argument are set in `actionname` and `argument` respectively.

4.2.3. Stack of Tasks

The Stack of Tasks (SoT) is a framework for efficiently implementing the Generalised Inverted Kinematics (GIK), first introduced in [20]. It was made to control redundant robots, ans is widely used in control of humanoid robots. The SoT framework is presented in [19]. It is a framework for organising multiple tasks that are to be completed at the same time by defining the relation between them. This sections is a summary of the work done in [19] where SoT is presented.

Task Definition and Handling
To explain the SoT framework several equations are needed. All equations and their corresponding variable explanation in this section is taken from [19].

A task in SoT is defined by the following formula:

$$\boldsymbol{e_i} = \boldsymbol{s_i} - \boldsymbol{s_i^*} \tag{4.16}$$

where $\boldsymbol{e_i}$ is the task, $\boldsymbol{s_i}$ is the current value of a feature, which will be explained later, and $\boldsymbol{s_i^*}$ is the desired value of the same feature. The task Jacobian $\boldsymbol{J}_i$ is found in the following formula:

$$\dot{\boldsymbol{e}}_i = \frac{\partial \boldsymbol{e_i}}{\partial \boldsymbol{q}} = \boldsymbol{J}_i \dot{\boldsymbol{q}} \tag{4.17}$$

where $\boldsymbol{q}$ is a vector containing the joint positions of the robot. A desired motion in task space is denoted $\dot{\boldsymbol{e}}_i^*$. By rewriting (4.17), it is possible to find the joint velocities $\dot{\boldsymbol{q}}$ required to execute this desired motion:

$$\dot{\boldsymbol{q}}_i = \boldsymbol{J_i}^+ \dot{\boldsymbol{e}}_i^* \tag{4.18}$$

where $\boldsymbol{J_i}^+$ is the pseudo-inverse of the task Jacobian as explained in Section 2.8.1. The desired motion $\dot{\boldsymbol{e}}_i^*$ is constrained by the following differential equation.

$$\dot{\boldsymbol{e}}_i^* = -\lambda \boldsymbol{e_i} \tag{4.19}$$

where lambda is a feature gain determining the magnitude of $\dot{\boldsymbol{e}}_i^*$. Inserting (4.19) into (4.18) will yield the following control law:

$$\dot{\boldsymbol{q}}_i = -\lambda \boldsymbol{J_i}^+ \boldsymbol{e_i} \tag{4.20}$$

Now a formula to calculate the task Jacobian $\boldsymbol{J_i}$ is required. For this definition a relation between the velocity of a point on the robot $\boldsymbol{v}$, and the value of a feature is needed. This can be written as:

$$\dot{\boldsymbol{s}}_i = \boldsymbol{L}_{s_i} \boldsymbol{v} \tag{4.21}$$

where $\boldsymbol{L}_{s_i}$ is called the interaction matrix which is used in visual servoing. It is now possible to define the task Jacobian $\boldsymbol{J_i}$ as:

$$\boldsymbol{J_i} = \boldsymbol{L}_{s_i} \boldsymbol{M} \boldsymbol{J_q} \tag{4.22}$$

where $\boldsymbol{J_q}$ is the joint Jacobian, and $\boldsymbol{M}$ is a matrix representing the velocity $\boldsymbol{v}$ from the joint Jacobian.

As with the other constraint-based solutions presented previously in this paper, SoT uses task prioritisation to deal with redundancy in the robot system. To explain redundancy handling in SoT, imagine n tasks defined as $(\dot{\boldsymbol{e}}_\mathbf{1}, \dot{\boldsymbol{J}}_\mathbf{1}), (\dot{\boldsymbol{e}}_\mathbf{2}, \dot{\boldsymbol{J}}_\mathbf{2}), ..., (\dot{\boldsymbol{e}}_\boldsymbol{n}, \boldsymbol{J_n})$. The prioritisation is defined such that task one is the most prioritised, and task n is the least prioritised. A task should never disturb a task with higher priority than itself. For computing joint velocities, a recursive computation presented in [31] has been used. In [19] the recursive computation is written as:

$$\dot{\boldsymbol{q}}_\mathbf{0} = 0 \tag{4.23}$$

$$\dot{\boldsymbol{q}}_\boldsymbol{i} = \dot{\boldsymbol{q}}_{\boldsymbol{i-1}} + (\boldsymbol{J_i} \boldsymbol{P}_{\boldsymbol{i-1}}^{\boldsymbol{A}})^+ (\dot{\boldsymbol{e}}_i - \boldsymbol{J_i} \dot{\boldsymbol{q}}_{\boldsymbol{i-1}}), \quad i = 1...n \tag{4.24}$$

The null-space projector of the augmented Jacobian $\boldsymbol{J}_i^{\boldsymbol{A}} = (\boldsymbol{J_1}, ..., \boldsymbol{J_n})$ is denoted

$\boldsymbol{P_i^A}$, and can be calculated as:

$$\boldsymbol{P_i^A} = \boldsymbol{P_{i-1}^A} - (\boldsymbol{J_i P_{i-1}^A})^{+} \boldsymbol{J_i P_{i-1}^A} \tag{4.25}$$

This recursive method use the joint velocities defined by the previous higher priority tasks, and adds joint velocities from the current task within their null-space. By computing joint velocities of a task within the null-space of all higher priority task it is guaranteed that a task will never disrupt another task with higher priority. After computing the joint velocities for all n tasks, the joint velocity vector $\boldsymbol{\dot{q}_n}$ can be sent to the controller.

Software

The implementation of SoT is based around computational nodes called entities. They are able to consume an input and give an output based on what it is set to do. The nodes are connected to each other through the inputs an outputs. All the entities and connections can be visualised through a graph like the one shown in Figure 4.10. This figure is taken from [19] and shows the entities and their connections for controlling a humanoid robot. Entities can also be connected with more complex relations like prioritisation.

To manage the SoT framework a scripting interface has been developed. This is an abstraction that allows the user to control the framework using a selection of basic commands. Examples of basic commands are the `new` command to create new entities and the `plug` command to connect entities together. More script functionalities can be loaded using plugins. As mentioned, the SoT framework consists of entities that communicate with each other. Features, tasks and SoT entities will now be explained.

A feature entity is is designed to give the task entity an input of how close it is to be completed. The input to a feature is data from the robot system like e.g. sensor data. It outputs the vectors $\boldsymbol{s}_i$ and $\boldsymbol{s}_i*$ along with the task Jacobian $\boldsymbol{J}_i$. The output from the feature entity is then sent to a corresponding task entity.

The task entities represent the tasks as defined in (4.16). They take input form the features and computes joint velocities using the control law defined in (4.20). After joint velocities are calculated for each task, the SoT entity handles task prioritisation.

The SoT entity is given a stack of different tasks each with their own prioritisation and joint velocities. The final joint velocities are then calculated using the recursive calculations shown in (4.24).

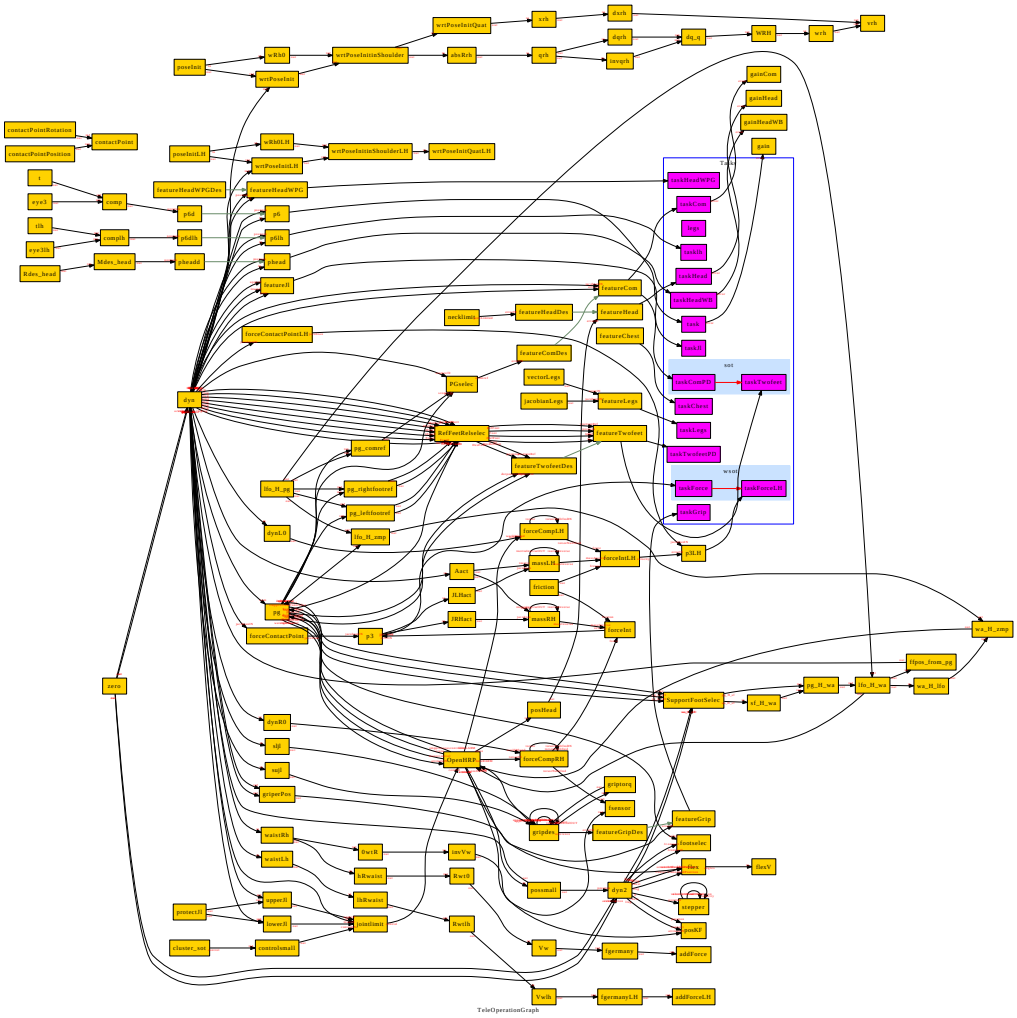


Figure 4.10.: A visualisation of the SoT framework. Figure from [19].

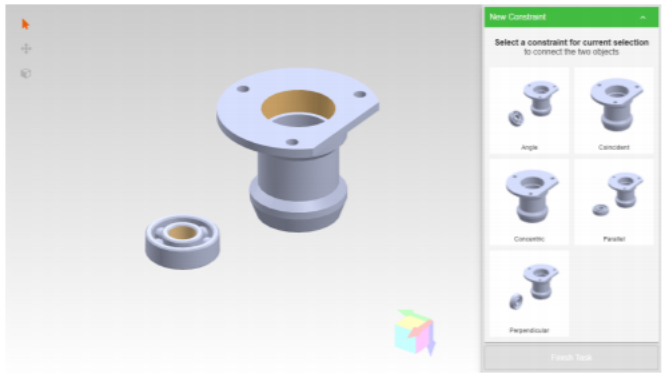


Figure 4.11.: The GUI made for easy task-constraint definitions in [34]. Figure from [34].

4.3. Additional Frameworks

There are several other frameworks for constraint-based robot programming than the three already presented in Section 4.2. This section will briefly mention some of the other existing frameworks.

In [33] and [34] a framework for a more intuitive approach to constraint-based robot programming is presented, with a goal of making it more available for "non-expert users". The approach is based on defining the constraints of task in a CAD-model of the workpieces. In addition to the task constraints the user can define constraints on the robot itself, and constraints imposed on the robot by the surrounding environment. All constraints are then run through a solver where a satisfactory robot movement is found. This framework also supports prioritisation of constraints. Figure 4.11 from [34] shows the GUI developed for easy definition of constraints on the task. The framework was further developed in [35] where a constraint solver with better runtimes was presented.

In [13] a constraint-based framework for controlling human-like robots is presented. Human motion is defined as a series of sub-tasks that are executed with different priority, this is the basis for the framework. While a task of high priority is executed, lower priority tasks are allowed to be performed as long as they do not interrupt the higher priority task. This is similar to most prioritisation seen in constraint-based robot programming.

Another framework, "task Space Inverse Dynamics", can be found in [10]. This is a framework for multiple prioritised tasks, using force/torque control. It was created to fill the gap for a framework yielding optimal solutions to secondary tasks that would still be computationally effective. The solution was to separate

the robot kinematics and robot dynamics, which yielded a more effective algorithm for computation. Like the other frameworks it allows completion of high priority tasks with lower priority tasks running in the background.

A different goal using constraint-based programming is shown in [15] where the goal is to program industrial robots for use in cooperation with humans. The hierarchical programming framework where tasks can be programmed with different priorities enables the safety of the human to always be of highest priority. This way the robot is able to execute tasks while at all times executing collision avoidance with the human. This was done by the use of 3D-vision. Three different industrial settings were a person collaborates with a robot were presented.

Lastly [25] presents a methodology for restricting the movements of objects in relation to each other using geometrical constraints. Additionally a geometric constraint-solver is presented, where the rotational and transnational components are separated and solved individually. This methodology and solver can be used to describe robot tasks.

4.4. Robotic Welding using Constraint-Based Robot Programming

The application of constraint-based robot programming in robotic welding is not widespread, but some research has been done in the field. This section will present this work.

In [36] a methodology for combining CAD and constraint-based programming to generate robot trajectories is presented. Seam tracking in robotic welding is brought up as an example for one type of task that can be defined using this methodology. Point and seam welding is also used as an example for applications of the framework in [33]. In addition, seam welding is used as an example for the improved geometric solver in [35].

An algorithm is presented in [3] where constraints are used to coordinate a seven-axis robot along with a two axis work bench to perform a welding operation on a work piece. This is done in a similar manner to recent work in the field where the task is divided into multiple sub-tasks. The difference occurs in the application of constraints. While more recent work uses multiple prioritised constraints, [3] applies three global constraints with equal priority that all subtasks needs to follow. The goal of the constraints in this case was to avoid singular joint positions and to keep the weld seam horizontal.

The application of constraint-based robot programming in robotic welding will be discussed further in Chapter 6.

Chapter 5.

Results

Results are divided in two sections. The first section will present the preparatory work that has been done to enable work on constraint-based robot programming with robotic welding. The second section will present the results obtained from the interviews conduced in collaboration with Thea Holmedal.

5.1. Preparations for Further Work

The robot cell available for further work on this subject includes a YASKAWA Motoman GP25-12 equipped with a Fronius TPS 400i welding system. This is a MIG/MAG type welding gun, with support for different welding methods like CMT. A two DOF work table for the workpiece is also present in the cell. The cell with all components can be seen in Figure 5.1.

A simplified model of the GP25-12 has been made to easily identify screw axes and relevant joint lengths. It is based on the technical drawing taken from [38] which can be found in Appendix B. The simplified model with a fixed frame {s} and a body-frame {b} can be seen in Figure 5.2.

When using the Python library from [18], it is useful to know the zero-position of the robot and the screw axis in both body and space frame. Using the drawing as reference and applying the theory from Section 2.7 the following matrices can be found:

$$\boldsymbol{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1.332 \\ 0 & 0 & 0 & 1.465 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

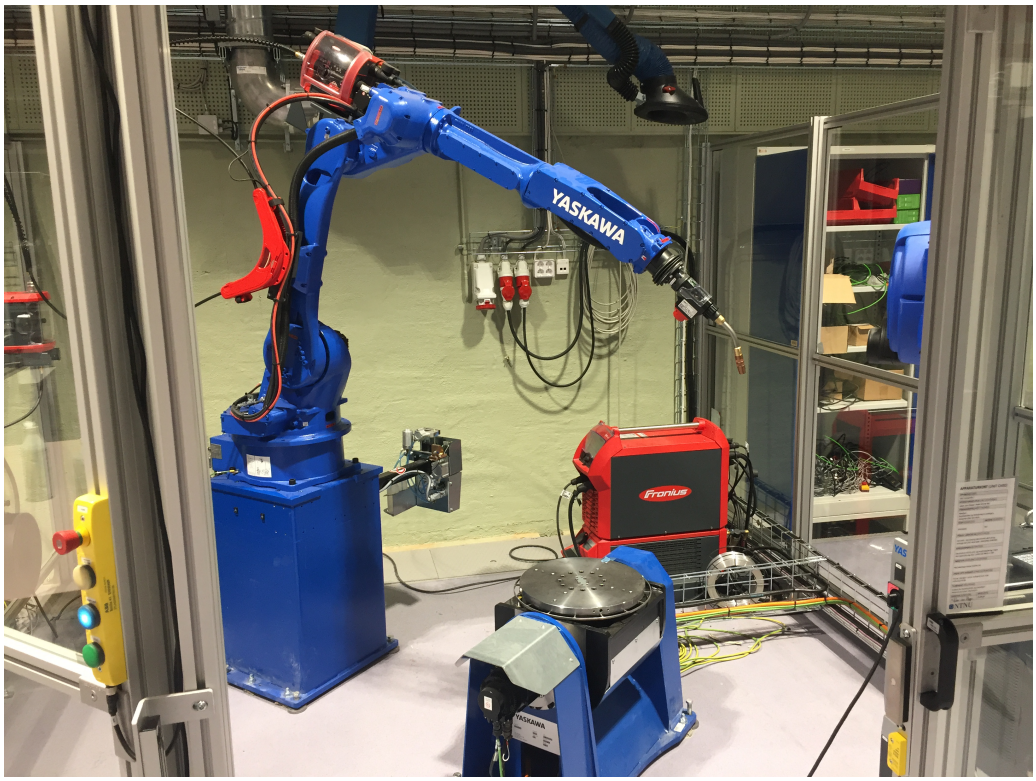


Figure 5.1.: The robot cell available at MTP, consisting of a YASKAWA Motoman GP25-12 equipped with a Fronius TPS 400i welding system. A two DOF working table is also available.

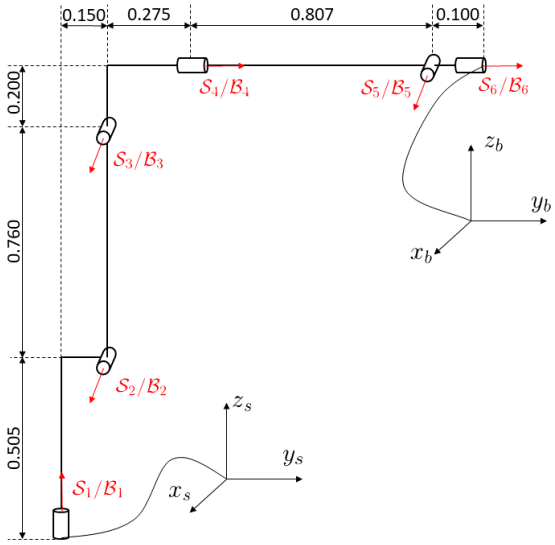


Figure 5.2.: Simplified model of the YASKAWA Motoman GP25-12, units are given in meters.

$$\boldsymbol{\mathcal{S}} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1.465 & 0 & -1.465 \\ 0 & 0.505 & 1.265 & 0 & 1.465 & 0 \\ 0 & -0.150 & -0.150 & 0 & -1.232 & 0 \end{bmatrix} \tag{5.2}$$

$$\boldsymbol{\mathcal{B}} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ -1.332 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.960 & -0.200 & 0 & 0 & 0 \\ 0 & 1.182 & 1.182 & 0 & 0.100 & 0 \end{bmatrix} \tag{5.3}$$

Where $\boldsymbol{M}$ is the zero position, $\boldsymbol{\mathcal{S}}$ is a matrix where each column represents a screw axis in space frame and $\boldsymbol{\mathcal{B}}$ is a matrix where each column represents a screw axis in body frame. The tool-tip frame is of interest as this is the tip of the welding gun. A constant homogeneous transformation matrix $\boldsymbol{T}_{bt}$ is used to represent the displacement of the the tool frame {t} with respect to the body frame {b}. Form the teach pendant the following information was found:

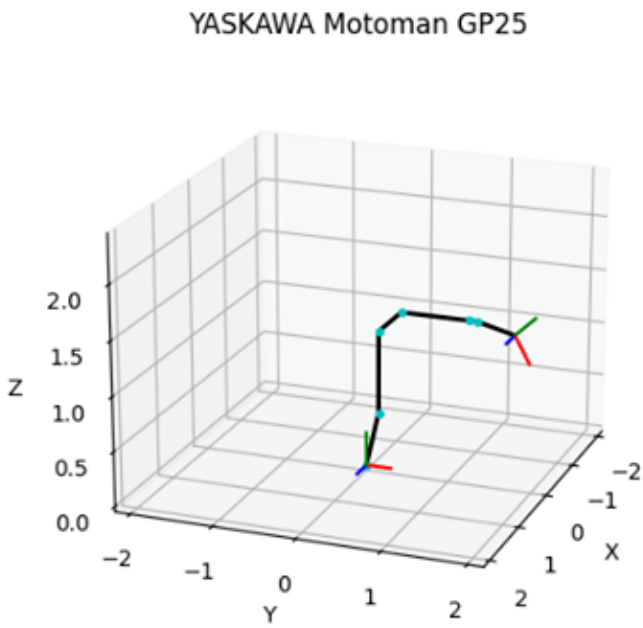


Figure 5.3.: Output form the robot visualiser showing the robot in zero-position.

$$\boldsymbol{T}_{bt} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(54°) & -\sin(54°) & 0.450 \\ 0 & \sin(54°) & \cos(54°) & -0.084 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

Using these matrices and the Python library from [18] a robot visualiser was made based on the PoE formulation. The purpose for this visualiser is to be able to check joint configurations without the need for the physical robot. Figure 5.3 shows the output from the visualiser when then the robot is set to zero-position.

5.2. Robotic Welding in the Norwegian Industry

This section will present the answers obtained from the interviews with Norwegian companies working with robotic welding. Three different subjects where interviewed, with an average relevant working experience of approximately four

years. The interview templates can be seen in Appendix A. This section will present each question of the interview along with the answers.

How do you program welding robots?
For simple welding operations, online programming using the teach pendant was the most common solution. The programming language provided by the manufacturer is used to generate welding paths based on saved points in the program. For one-of-a-kind tasks or tasks for lower production volumes, teach pendant programming also seems to be the preferred solution as it lets the user quickly program the robot.

For more complex tasks or bigger robot cells, offline programming using a CAD-model of the robot cell seems to be more popular. The advantage of being able to simulate the programs in the CAD-model, as well as not having to take robots out of production to program them is the reason offline programming is used. As with online programming, programs provided by the robot manufacturers are used in offline programming as well.

Some of the interview subjects works with delivering robot systems to customers in Norway. It then became clear that robot programming is something the customers themselves want to learn. This enables the customer to reprogram the robots to execute different tasks when production changes.

Do you use any sensors along with the robots?
All subjects use sensors with their welding robots, but not necessarily the same ones. In the industry the most common sensor proved to be through-arc sensing for seam tracking. One of the subjects also use touch sensing to locate the orientation of the workpiece to be welded.

Cameras are also in use. One subject use 3D-cameras with a projected laser grid to map the part so the welding process can be adjusted before it starts. In this case the camera is not used to track the groove during the welding process, as this will require a much more expensive camera. This is due to the extensive filtering that has to be done to execute seam tracking through the light from the welding arc. Despite the cost, another subject use 3D-cameras with laser projections for seam tracking and geometry detection.

Which welding methods does your robot use?
From the answers obtained it is clear that MIG/MAG is the most used welding method, as this is the cheapest and most robust. MIG/MAG is also a versatile welding method due to the advancements done in power supplies. With advanced power supplies it is possible to use e.g. CMT. More advanced welding methods will in turn enable the MIG/MAG welding guns to weld more metals like aluminium and titanium, as well as welding thin metal plates that requires low heat transfer.

TIG is also used in the Norwegian industry when the task requires this kind of welding quality. This is typical for offshore products. TIG is a more cumbersome welding method to use in robotic welding as it requires coordination of the filler metal and the electrode. This will for example limit the motions of the robot by one DOF as the filler metal always has to point in the direction of the weld.

The subjects also gave examples of other welding methods that are common in the robot industry, but not necessarily in Norway. These methods are FSW, RSW and laser welding which are all typically used in the automotive industry. As this type of industry is not widespread in Norway, they are less popular.

What is the biggest challenge you face when programming welding robots?
There is no clear trend in the answers to this question. One subject mentioned hardware problems with the robot e.g. I/O problems and malfunctioning sensors. Complex welding tasks where also brought up, as they might require more programming skills to implement than the robot programmer has.

For another subject the biggest problem is the different orientation of parts in a production line. It is then a challenge to adapt the welding path to be correct each time. Through-arc sensing was brought up as a possible solution to this problem although it was not implemented at the time.

The last subject brought up user-friendliness. It is a challenge to make the programming interface user friendly enough. The goal is to make robot programming as intuitive and easy for the operator as possible. Welding parameters are also challenging to set, and a welding engineer is often used to give guidance to the settings on the welding apparatus.

To what extent are you able to automate the welding process as a whole?
It is usually possible to reach a high degree of automation. One customer mentioned that if it is not possible to fully automate the process, it is not any point to automate it in the first place. On the other hand another subject usually based the automation process on 80% automation, 20% manual work. In processes where the weld would leave a lot of spatter or slag, it is usually done manually from the start as the slag has to be ground off manually anyways.

In most cases design of the production line determines the possible degree of automation. Parts can be oriented in optimal ways, and welding seams can be placed strategically on the product.

Chapter 6.

Discussion

This chapter will combine theory with the results to discuss the potential benefits and challenges related to the implementation of constraint-based robot programming for welding robots in the Norwegian industry.

Robot types
As seen in Section 3.3, many industrial robots have six-axis. This is satisfactory for movement with six DOF, which yields versatility and enables execution of most tasks. The use of robots with more than six-axis requires a way of handling redundancy, as it will always be present. With constraint-based robot programming it is possible to set lower priority constraints on the robot to handle redundancy in a meaningful way. This can be to avoid singularities, minimise energy consumption or minimise joint movements. The redundancy can also be used for collision avoidance, which in turn can let the robot operate in tighter spaces than a more standard six-axis robot. Being able to operate in tight spaces might also make robotic welding available in operations previously too complex or in a too tight place for automation with six-axis robots.

Simplifying the use of more than six-axis robots will also be beneficial for welding operations like TIG-welding, where a filler metal has to be added separately. As mentioned in the interviews, TIG welding reduces the DOF of a robot by one, since the filler metal always has to be supplied in front of the arc. By using a robot with more than six axes, more flexible robot movement can still be achieved in spite of the reduced DOF. This could be relevant for the Norwegian industry due to TIG requirements in the offshore sector.

Easier control of redundant robots could also lead to the use of highly redundant robots in robotic welding. These are snake like robots with many joints. These kind of robots are highly flexible, and would be able to accomplish welding operations deemed too complex for the traditional six-axis robot.

Welding methods
All welding methods presented in Section 3.1 should be compatible with constraint-based robot programming as long as the welding task can be expressed by task constraints. As MIG is the most popular welding method in Norway, this should be the first welding method to be implemented with constraint-based robot programming.

Assuming constraint-based robot programming will lead to a higher degree of automation due to more flexible robots being used, it will be of interest to utilise welding method where no post processing is required. For the Norwegian industry, this would mainly be MIG/MAG and TIG welding.

TIG welding has no splatter and uses no flux. If automating TIG is made easier by constraint-based robot programming, it could end up as a more popular welding method than it is today. Specialised MIG processes like CMT would also be of interest due limited splatter and versatile welding opportunities regarding types of metal an thickness of the workpieces.

Welding sensors
In the industry, cost is an important factor. It would therefore be beneficial for welding robots work with only through-arc and touch sensing, as these are the cheapest and most available. The accuracy of the sensors would then have to be considered. A sensor for a constraint-based robot program would need to be able to measure all defined constraints. Through-arc and touch sensing might be satisfactory for assessing task-constraints while welding, but will be not work when the arc is inactive. For the robot to be able to evaluate constraints without an active arc, the use of 3D-cameras would be required.

Another issue with through-arc sensing is the more advanced arc welding processes where changing the current and voltage is part of the process. Through-arc sensing assumes a constant voltage so that the current can be used to measure distance. For a process like CMT the current and voltage is changed intentionally as a part of the welding process. This will interrupt the through-arc sensing process rendering it useless.

For other welding methods like SRW, FSW or laser welding, 3D-cameras would also be a viable sensor choice. Through arc-sensing is not an option for these methods as they are not arc welding operations.

The main drawback of using 3D-cameras are the cost. A basic camera for detecting the orientation of parts is cheaper than the ones used for seam tracking, but both are more expensive options than through arc sensing. In a case where CMT is deployed for less post-production, the user would have to compare the cost of using a 3D-camera to the cost of post production when using through-arc sensing and traditional MIG/MAG.

Framework
As seen in Chapter 4 there are several frameworks for constraint-based programming, and many of them are viable options for a robotic welding implementation. Seeing that robotic welding is commonly executed by industrial robots, the frameworks made for humanoid robots, like SoT, might not be ideal for smaller robot cells. For large robot cells with several robots resulting in a high DOF, these kind of frameworks would be more viable options.

The degree of development for a framework would also an important factor when implementing constraint-based robot programming to welding robots. Based on this criteria, the work done in [33], [34] and [35] could be a viable alternative, but eTaSL/eTC would probably provide an even more developed framework due to its developed script for interfacing with the framework. It has already been used for assembly tasks [16] [17], and could probably be used efficiently in robotic welding as well.

The problem using eTaSL/eTC is the non-intuitive programming interface, which will require programming skills to be able to operate. In an industrial setting, users with varying programming knowledge should be able to program the robots. By implementing a GUI similar to the one seen in [34], eTC/eTaSL could be made more available for non-programmers which would yield more flexibility in who is able to reprogram the welding robots.

Area of Use
Constraint-based robot programming will simplify the process of programming complex tasks, and might reduce the required programming time than what is currently required. For simple tasks it is seen from the results that teach-pendant programming is the most popular. If constraint-based robot programming is to be used in this kind of simple tasks, it would have to be implemented in the teach-pendant. To compete with the current teach-pendant programming method it would have to be fast and user friendly, which might be possible with a well developed constraint-based programming framework. The operator should be able to define the constraints on the pendant, which are sent to the solver to calculate the robot trajectory. Standard low priority tasks like singularity avoidance and minimal joint movements could also be saved to the pendant for easy access.

The main use for constraint-based robot programming would still be that of programming complex tasks. This is usually done offline and would not require the same level of fast and intuitive programming seen in teach-pendant programming, although it should make the process easier and less time consuming than it is today.

Norwegian industry does not have the same level of manufacturing mass production as is usually associated with e.g. the automobile industry. Smaller batch

sizes often means more specialised and complex products. This can be challenging for robot programmers as the robots will have to be reprogrammed often. With constraint-based robot programming, this process could be made more efficient even for more complex welding operations. This would lead to less down time due to robot setup, which in turn could enable robotic welding in productions where it was previously not possible due to time consumption. Easier programming of complex tasks would have the same effect, meaning a wider array of products can be welded by robots.

Chapter 7.

Conclusion

A literature review on state of the art frameworks for constraint-based robot programming and welding technologies has been conducted, along with interviews of relevant subjects in the Norwegian industry. Based on this work discussions have been made on the potential of introducing constraint-based programming. The discussion made the following conclusions:

- Easier control of redundant industrial robots using constraint-based robot programming could lead to the use of more flexible robots which will enable execution of welding tasks previously too complex for robotic welding.
- All welding methods could be compatible with constraint-based robot programming. In the Norwegian industry the focus should be on MIG welding. TIG welding could also be more popular due to more flexible robots which will simplify TIG welding automation.
- 3D cameras would be required for evaluating the constraints when running a robot with constraint-based robot programming, as through-arc sensing only works with an active arc and not on welding methods with changing current like CMT. This is a welding method that could be desirable to use.
- Constraint-based robot programming can make programming complex tasks faster and easier, enabling more tasks to be automated.
- A well developed framework for constraint-based robot programming could be implemented in the teach-pendant for online programming, but the main area of use would be offline programming of complex tasks. It is concluded that eTC/eTaSL is a viable framework due to it being well developed.

In addition to the discussion, a simple model and visualiser was made for the welding robot available at the Department of Mechanical and Industrial Engineering (MTP) at the Norwegian University of Science and Technology (NTNU).

7.1. Further work

Further work can be done in the process of mapping the current situation for robotic welding and the need for constraint-based robot programming. This report has interviewed three subjects in Norway. More subjects in Norway as well as other countries, where the market is different, should be included in this research.

A proof of concept has to be developed where a framework for constraint-based robot programming is applied to a welding robot. Simple welding tests should be done to test the accuracy of the implementation compared to more traditional robot programming methods.

Given a working proof of concept, a framework for easy task specification must be implemented to make programming easy and intuitive. Work should also be done to see what kind of sensors and welding methods that would be optimal for constraint-based robot programming.

References

[1] ABB. *ABB IRB 1300*. 2020. URL: https://new.abb.com/products/robotics/industrial-robots/irb-1300 (visited on 11/21/2020).

[2] Erwin Aertbeliën and Joris De Schutter. "eTaSL / eTC : A constraint-based Task Specification Language and Robot Controller using Expression Graphs". In: ().

[3] Shaheen Ahmad and Shengwu Luo. "Coordinated motion control of multiple robotic devices for welding and redundancy coordination through constrained optimization in Cartesian space". In: *Proceedings. 1988 IEEE International Conference on Robotics and Automation*. IEEE. 1988, pp. 963–968.

[4] FANUC Robotics America. *Through Arc Seam Tracking (TAST)*. 2005.

[5] Geoffrey Biggs and Bruce MacDonald. "A survey of robot programming systems". In: *Proceedings of the Australasian conference on robotics and automation*. 2003, pp. 1–3.

[6] ABICOR BINZEL. *TH6x Optical Seam Tracking Sensor*. 2020. URL: https://www.binzel-abicor.com/US/eng/products/robotic-systems/seam-tracking-gas-control/seam-tracking-sensor-th6d/ (visited on 12/03/2020).

[7] E. Sahin Conkur and Rob Buckingham. "Clarifying the definition of redundancy as used in robotics". In: *Robotica* 15.5 (1997), pp. 583–586. ISSN: 02635747. DOI: 10.1017/S0263574797000672.

[8] Joris De Schutter, Tinne De Laet, Johan Rutgeerts, Wilm Decré, Ruben Smits, Erwin Aertbeliën, Kasper Claes, and Herman Bruyninckx. "Constraint-based Task Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty". In: *The International Journal of Robotics Research* 26.5 (2007), pp. 433–455. DOI: 10.1177/0278364907078091. URL: http://ijr.sagepub.com.

[9] Wilm Decŕe, Ruben Smits, Herman Bruyninckx, and Joris De Schutter. “Extending iTaSC to support inequality constraints and non-instantaneous task specification”. In: *Proceedings - IEEE International Conference on Robotics and Automation* (2009), pp. 964–971. ISSN: 10504729. DOI: 10.1109/ROBOT.2009.5152477.

[10] Andrea Del Prete, Francesco Nori, Giorgio Metta, and Lorenzo Natale. “Prioritized motion-force control of constrained fully-actuated robots: "task Space Inverse Dynamics"”. In: *Robotics and Autonomous Systems* 63.P1 (2015), pp. 150–157. ISSN: 09218890. DOI: 10.1016/j.robot.2014.08.016. arXiv: 1410.3863. URL: http://dx.doi.org/10.1016/j.robot.2014.08.016.

[11] FANUC. *FANUC ARC Mate 100iD*. 2020. URL: https://www.fanuc.eu/pl/en/robots/robot-filter-page/arc-welding/arcmate-100id?returnurl=https%5C%3A%5C%2F%5C%2Fwww.fanuc.eu%5C%2Fpl%5C%2Fen%5C%2Frobots%5C%2Frobot-filter-page%5C%23%5C%3Ft%5C%3D22a25ffda3b44d9293e9915737463cc2%5C%2C030001b7b73a4d45ac7b6acfd2170c2d (visited on 11/21/2020).

[12] Mikell P. Groover. *Principles of Modern Manufacturing*. 5th ed. John Wiley & Sons Singapore Pte. Ltd., 2013.

[13] O. KHATIB, L. SENTIS, J. PARK, and J. WARREN. “Whole-Body Dynamic Behavior and Control of Human-Like Robots”. In: *International Journal of Humanoid Robotics* 01.01 (2004), pp. 29–43. ISSN: 0219-8436. DOI: 10.1142/s0219843604000058.

[14] KUKA. *KUKA KR QUANTEC prime*. 2020. URL: https://www.kuka.com/en-se/products/robotics-systems/industrial-robots/kr-quantec-prime (visited on 11/21/2020).

[15] Claus Lenz, Markus Rickert, Giorgio Panin, and Alois Knoll. “Constraint task-based control in industrial settings”. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009* (2009), pp. 3058–3063. DOI: 10.1109/IROS.2009.5354631.

[16] Dag Lofthus. “From CAD-Assemblies to Constraint Based Robot Control”. MA thesis. Norwegian University of Science and Technology, 2019.

[17] Espen Lunden. “Force Control and Constraint-based Task Specification in Robotic Assembly”. MA thesis. Norwegian University of Science and Technology, 2020.

[18] Kevin M Lynch and Frank C Park. *Modern Robotics*. Cambridge University Press, 2017.

[19] Nicolas Mansard, Olivier Stasse, Paul Evrard, and Abderrahmane Kheddar. "A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks". In: *2009 International Conference on Advanced Robotics, ICAR 2009* (2009).

[20] Yoshihiko Nakamura and Hideo Hanafusa. "Optimal Redundancy Control of Robot Manipulators". In: *The International Journal of Robotics Research* 6.1 (1987), pp. 32–42. ISSN: 17413176. DOI: 10.1177/027836498700600103.

[21] OcorosWiki. *What is iTaSC?* 2020. URL: https://www.orocos.org/wiki/orocos/itasc-wiki/1-what-itasc (visited on 11/09/2020).

[22] Alex Owen-Hill. *What Is the Best Way to Program a Robot?* 2018. URL: https://robodk.com/blog/program-robot-tips/ (visited on 12/05/2020).

[23] J Norberto Pires, Altino Loureiro, and Gunnar Bölmsjo. *Welding robots: technology, system issues and application*. Springer Science & Business Media, 2006.

[24] Matthias Radestock and Susan Eisenbach. "Coordination in evolving systems". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 1161 (1996), pp. 162–176. ISSN: 16113349. DOI: 10.1007/3-540-61842-2_34.

[25] Adolfo Rodríguez, Luis Basañez, and Enric Celaya. "A relational positioning methodology for robot task specification and execution". In: *IEEE Transactions on Robotics* 24.3 (2008), pp. 600–611. ISSN: 15523098. DOI: 10.1109/TRO.2008.924263.

[26] Enric Rodriguez-Carbonell. *Intro to Constraint Programming*. 2020. URL: https://www.cs.upc.edu/~erodri/webpage/cps/theory/cp/intro/slides.pdf (visited on 12/14/2020).

[27] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[28] Amruta Rout, B. B.V.L. Deepak, and B. B. Biswal. "Advances in weld seam tracking techniques for robotic welding: A review". In: *Robotics and Computer-Integrated Manufacturing* 56.June 2017 (2019), pp. 12–37. ISSN: 07365845. DOI: 10.1016/j.rcim.2018.08.003. URL: https://doi.org/10.1016/j.rcim.2018.08.003.

[29] Claude Samson, Bernard Espiau, and Michel Le Borgne. *Robot control: the task function approach*. Oxford University Press, Inc., 1991.

[30] S. Selvi, A. Vishvaksenan, and E. Rajasekar. *Cold metal transfer (CMT) technology - An overview*. 2018. DOI: 10.1016/j.dt.2017.08.002.

[31] B. Siciliano and J.-J.E. Slotine. “A general framework for managing multiple tasks in highly redundant robotic systems”. In: 2.October (1991), 1211–1216 vol.2. DOI: 10.1109/icar.1991.240390.

[32] Bulgarian Software Solutions. *Seam Finding and Tracking camera for Welding robot*. 2020. URL: http://www.bssbg.com/products/seam-finding-and-tracking-camera-for-welding-robot (visited on 12/03/2020).

[33] Nikhil Somani, Andre Gaschler, Markus Rickert, Alexander Perzylo, and Alois Knoll. “Constraint-based task programming with CAD semantics: From intuitive specification to real-time control”. In: *IEEE International Conference on Intelligent Robots and Systems* 2015-Decem (2015), pp. 2854–2859. ISSN: 21530866. DOI: 10.1109/IROS.2015.7353770.

[34] Nikhil Somani, Markus Rickert, Andre Gaschler, Caixia Cai, Alexander Perzylo, and Alois Knoll. “Task level robot programming using prioritized non-linear inequality constraints”. In: *IEEE International Conference on Intelligent Robots and Systems* 2016-November (2016), pp. 430–437. ISSN: 21530866. DOI: 10.1109/IROS.2016.7759090.

[35] Nikhil Somani, Markus Rickert, and Alois Knoll. “An Exact Solver for Geometric Constraints with Inequalities”. In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 1148–1155. ISSN: 23773766. DOI: 10.1109/LRA.2017.2655113.

[36] G. C. Vosniakos and A. Chronopoulos. “Industrial robot path planning in a constraint-based computer-aided design and kinematic analysis environment”. In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 223.5 (2009), pp. 523–533. ISSN: 09544054. DOI: 10.1243/09544054JEM1234.

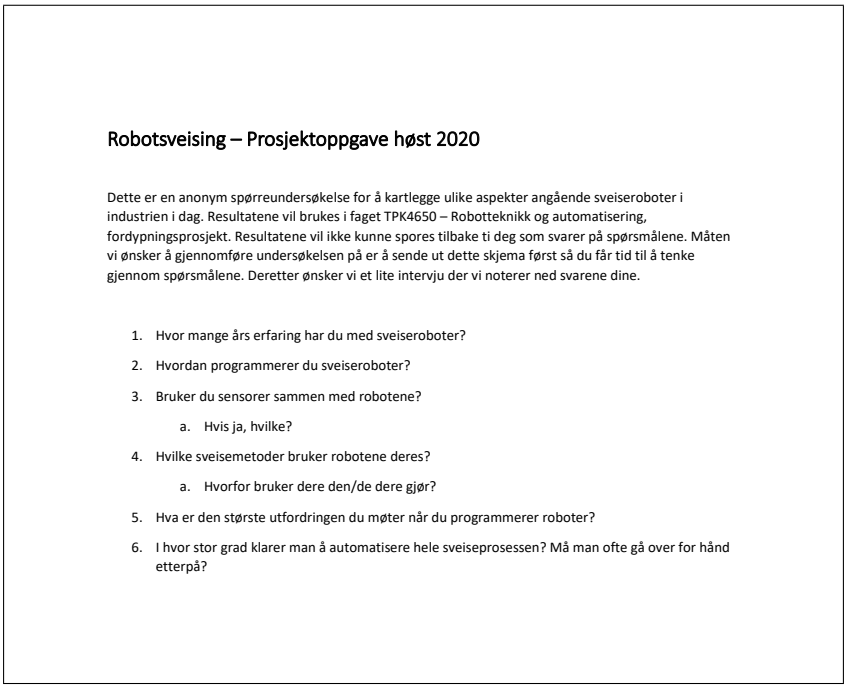
[37] YASKAWA. *YASKAWA AR900*. 2020. URL: https://www.yaskawa.eu.com/products/robots/welding-cutting/productdetail/product/ar900_733 (visited on 11/21/2020).

[38] YASKAWA. *YASKAWA Motoman GP25-12*. 2020. URL: https://www.yaskawa.eu.com/products/robots/handling-mounting/productdetail/product/gp25_699 (visited on 12/08/2020).

Appendix A.

Interview Templates

A.1. Norwegian Interview Template



Robotsveising – Prosjektoppgave høst 2020

Dette er en anonym spørreundersøkelse for å kartlegge ulike aspekter angående sveiseroboter i industrien i dag. Resultatene vil brukes i faget TPK4650 – Robotteknikk og automatisering, fordypningsprosjekt. Resultatene vil ikke kunne spores tilbake ti deg som svarer på spørsmålene. Måten vi ønsker å gjennomføre undersøkelsen på er å sende ut dette skjema først så du får tid til å tenke gjennom spørsmålene. Deretter ønsker vi et lite intervju der vi noterer ned svarene dine.

1. Hvor mange års erfaring har du med sveiseroboter?
2. Hvordan programmerer du sveiseroboter?
3. Bruker du sensorer sammen med robotene?
    a. Hvis ja, hvilke?
4. Hvilke sveisemetoder bruker robotene deres?
    a. Hvorfor bruker dere den/de dere gjør?
5. Hva er den største utfordringen du møter når du programmerer roboter?
6. I hvor stor grad klarer man å automatisere hele sveiseprosessen? Må man ofte gå over for hånd etterpå?

Figure A.1.: Interview template in Norwegian.

A.2. English Interview Template

Robotic Welding – Specialization project fall 2020

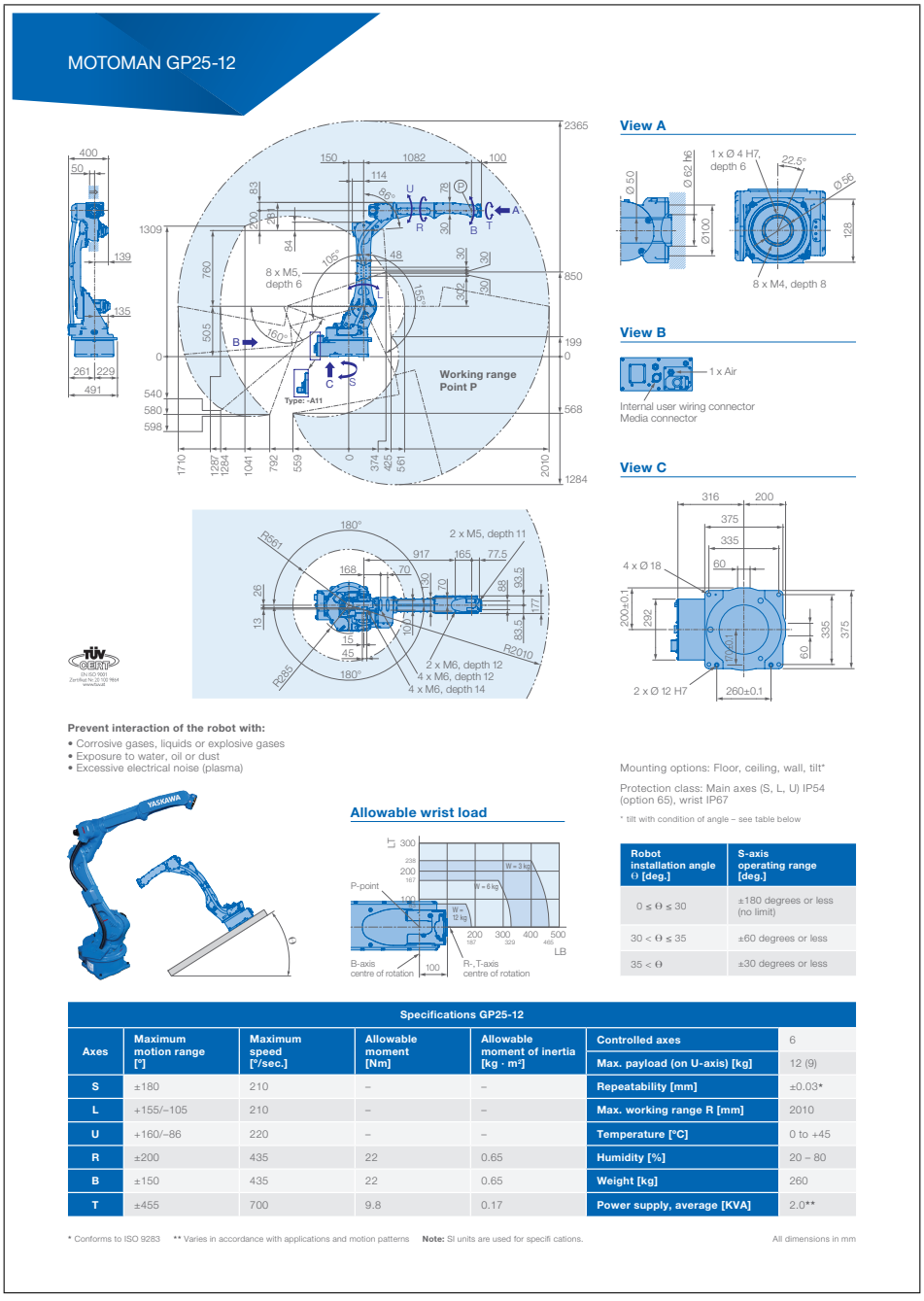
This is an anonymous survey to map different aspects regarding welding robots in today's industry. The results will be used in the subject TPK4650 – Robotics and Automation, Specialization Project. The results cannot be traced back to the one who answers the questions. This document contains the questions in the survey. We want you to go through the questions as a preparation. Then we wish to conduct an interview where we ask the questions and take notes of your answer.

1. How many years of experience do you have with welding robots?
2. How do you program welding robots?
3. Do you use any sensors along with the robots?
    a. If yes, which ones?
4. Which welding methods does your robots use?
    a. Why do you use this specific method?
5. What is the biggest challenge you face when programming welding robots?
6. To what extent are you able to automate the welding process as a whole? Is there often a need for manual welding where the robot cannot reach?

Figure A.2.: Interview template in English.

Appendix B.

Motoman GP25-12



Robot installation angle Θ [deg.]	S-axis operating range [deg.]
0 ≤ Θ ≤ 30	±180 degrees or less (no limit)
30 < Θ ≤ 35	±60 degrees or less
35 < Θ	±30 degrees or less

Specifications GP25-12						
Axes	Maximum motion range [°]	Maximum speed [°/sec.]	Allowable moment [Nm]	Allowable moment of inertia [kg · m²]	Controlled axes	6
					Max. payload (on U-axis) [kg]	12 (9)
S	±180	210	–	–	Repeatability [mm]	±0.03*
L	+155/–105	210	–	–	Max. working range R [mm]	2010
U	+160/–86	220	–	–	Temperature [°C]	0 to +45
R	±200	435	22	0.65	Humidity [%]	20 – 80
B	±150	435	22	0.65	Weight [kg]	260
T	±455	700	9.8	0.17	Power supply, average [KVA]	2.0**

* Conforms to ISO 9283 ** Varies in accordance with applications and motion patterns **Note:** SI units are used for specifi cations. All dimensions in mm

Figure B.1.: Technical specifications for YASKAWA Motoman GP25-12. Figure from [38].