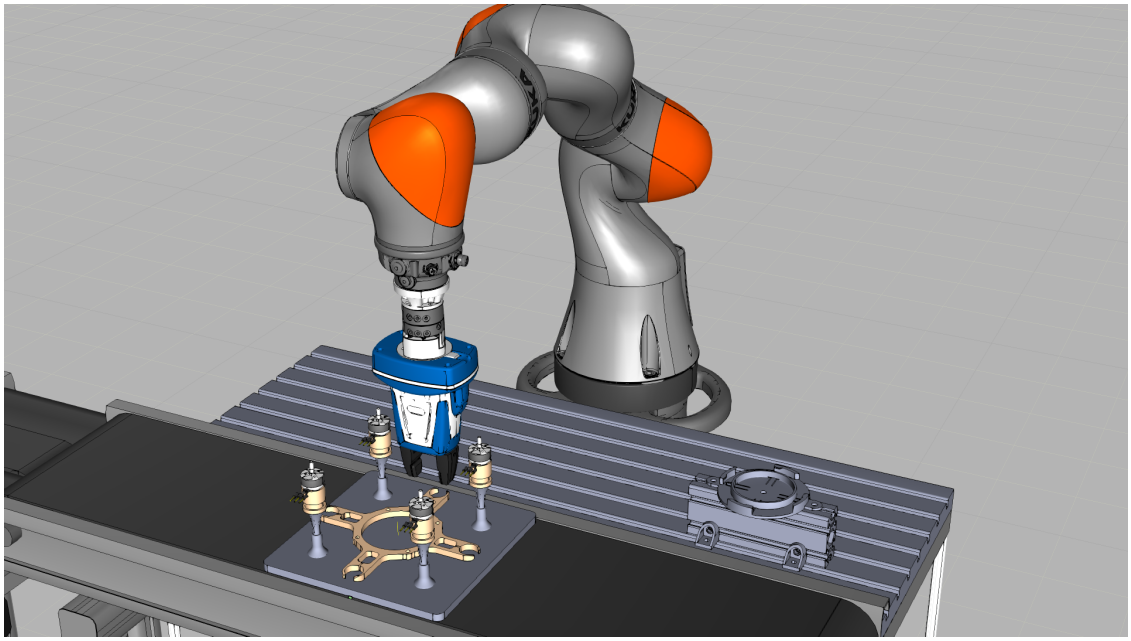
CHALMERS
UNIVERSITY OF TECHNOLOGY



# Virtual Commissioning of a drone assembly cell

Master's thesis in Production Engineering

VIVEK KOPPAL
CHETHAN SHIVARAJU

Department of Industrial and Materials Science
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

# Virtual Commissioning of a drone assembly cell

Performance testing of the KUKA iiwa robot and a comparative study of Virtual Commissioning process with Visual Components and Tecnomatix Process Simulate.

VIVEK KOPPAL
CHETHAN SHIVARAJU


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Industrial and Materials Science
*Division of Production Systems*
Chalmers University of Technology
Gothenburg, Sweden 2020

Virtual Commissioning of a drone assembly cell
Performance testing of the KUKA iiwa robot and a comparative study of Virtual Commissioning process with Visual Components and Tecnomatix Process Simulate
VIVEK KOPPAL, CHETHAN SHIVARAJU



Supervisor: Per Nyqvist, Department of Industrial and Materials Science
Examiner: Björn Johansson, Department of Industrial and Materials Science

Master's Thesis 2020
Department of Industrial and Materials Science
Division of Production Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Virtual model of the drone assembly cell created on Visual Components.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Virtual Commissioning of a drone assembly cell
Performance testing of the KUKA iiwa robot and a comparative study of Virtual Commissioning process with Visual Components and Tecnomatix Process Simulate
VIVEK KOPPAL
CHETHAN SHIVARAJU
Department of Industrial and Materials Science
Chalmers University of Technology

# Abstract

Stena industry innovation lab (SII-Lab) at Lindholmen, Göteborg is a research laboratory that houses a testbed within Production 2030 for sustainable production. A drone factory is one of the projects undertaken at SII-Lab. This master thesis project aimed at automating the assembly operation of drone frame chassis with drone motors using a 7-axis KUKA LBR iiwa robot adopting the Virtual Commissioning method. Virtual Commissioning method is used to evaluate the functionality, performance and safety of a production system. This method was used to simulate the assembly operation and link the programmable logic to validate the behaviour.

The project followed a methodology drafted from an earnest literature research on Virtual Commissioning process, that includes stages such as, data analysis, process planning, 3D modelling, simulation and logical control programming. An additional stage of physical implementation and verification aided in scrutinizing the process. The assembly operation was simulated using Visual Components and Tecnomatix Process Simulate. Further, physical station was setup by calibrating and programming the KUKA robot and testing for accuracy with reference to simulation applications. PLC programming for the system was prepared and tested to achieve Virtual Commissioning. Along with the process followed, the project aimed at answering research questions that are presented in the report. An additional research on force/torque analysis using the sensors available on the KUKA robot were conducted, to present possibilities in the drone assembly operation. The results of the project are presented and concluded that, Virtual Commissioning is an efficient method that can be adopted by manufacturing organizations for improved performance and cost saving solutions, without production stoppage or during setup of brownfield factories, involving process and technology improvements in the existing facility and greenfield factories, while establishing new production solutions and processes.

Keywords: Virtual Commissioning (VCom), KUKA iiwa robot, Tecnomatix Process Simulate (TPS), Visual Components (VC), KUKA Sunrise Workbench, Force/-Torque control, TIA Portal, PLCSIM.

# Acknowledgements

We would like that thank Åsa Fast-Berglund, Professor at the division of Production Systems, Department of Industrial and Materials Science, for providing us with the opportunity to conduct our experiments at Stena industry innovation lab. We also thank Sven Ekered, Research Engineer at the division of Productions Systems, Department of Industrial and Materials Science, for providing us with all the support during our project by sharing CAD files and 3D printing the parts numerous times and arranging to work on the SII-Lab laptop. We would like to express our gratitude to Björn Johansson, Professor at the division of Production Systems, Department of Industrial and Materials Science and the examiner of this Master thesis project, for supporting and encouraging us throughout the project duration.

Further, We would like to extend our deepest gratitude to our project supervisor, Per Nyqvist, Research Engineer at the division of Production Systems, Department Industrial and Materials Science, for his generous support, constant motivation and patience during the whole duration. We are in-debt of his support during our project cancellation at GKN Aerospace and helping us in setting up a new project at Stena industry innovation lab. We appreciate his swift response in providing us with a computer room and all required software applications immediately after information.

Lastly, We would like to express our deepest gratitude and love to our parents for their constant support during our graduate studies at Chalmers University and, for believing and guiding us throughout. The pandemic has taken us on an emotional roller-coaster and our family has helped us cope in these difficult times from 4000 miles apart. We would like dedicate this report to our families.

Vivek Koppal & Chethan Shivaraju, Gothenburg, August 2020

# Contents

# List of Figures

# List of Tables

# Acronyms

**DES** Discrete event simulation.
**DOF** Degrees of freedom.

**FBD** Function block diagram.

**HiL** Hardware in loop.
**HRC** human robot collaboration.

**iiwa** intelligent industrial work assistant.
**IL** Instruction list.

**LAD** Ladder diagrams.
**LBR** Leichtbauroboter (german for lightweight robot).
**LIN** Linear.

**PLC** Programmable logic controller.
**PnP** Plug n Play.
**PTP** Point to Point.

**RATE** Robotic Assembly Time Estimator.
**ROBEX** RObot Based Expert system.

**SFC** Sequential function chart.
**SII-Lab** Stena industry innovation lab.
**SiL** Software in loop.
**ST** Structured text.

**TCP** Tool centre point.
**TIA** totally integrated automation.
**TPS** Tecnomatix Process Simulate.

**VC** Visual Components.
**VCom** Virtual Commissioning.

# 1
# Introduction

This chapter provides a brief introduction to the thesis project. The chapter sheds light on the background, aim and limitations of the thesis project. Research questions to be answered through the thesis project are also provided.

## 1.1 Background

Manufacturers strive to achieve the ability to design and produce good quality products for sustenance and profits [1]. As the marketplace demands are rapidly changing, manufacturers need to remain competitive by continuously improving the production systems. Efficient prototyping of production systems seem important as improvement seeks high investments. Thus, a computer based environment to simulate and verify individual manufacturing processes is deemed essential and can be accomplished through virtual commissioning [1].

According to Metzner [2], Virtual Commissioning is an evaluation method of production system's functionality, performance and safety in the digital environment prior to physical implementation. The process involves using a digital twin of the production system. The simulated model is linked with programmable logic controllers to validate the behaviour [2]. This master thesis project aims to perform Virtual Commissioning of a drone assembly cell at the Stena industry innovation lab (SII-Lab) at Lindholmen, Gothenburg, Sweden.

Stena industry innovation lab (SII-Lab) is a research laboratory located in Sweden's largest science park at Lindholmen. SII-Lab is a testbed within Production 2030 - Sweden's innovation program for sustainable production. It is focusing on fast communication systems with 5G and collaborative robots, virtual and augmented reality techniques for final assembly [3]. Chalmers University is conducting various research projects in SII-Lab to test digitalized production of the future. A drone factory using collaborative robots is one of the projects under development. The drone factory is a realistic production system which includes products from various suppliers. The aim of this factory is to produce a drone assembly in order to illustrate the required hardware and software within SII-Lab [3]. The factory operates through material received from storage on conveyors and then are assembled by collaborative robots.

This master thesis primarily focuses on the assembly of motors on to drone chassis

with the KUKA iiwa 14kg robot. KUKA LBR iiwa 14kg is a light weight, sensitive and human robot collaboration (HRC) compatible robot with a payload of 14 kilograms capable of working on sensitive tasks [4]. Test drone components are considered in the execution of this master thesis project.

## 1.2 Project purpose and aim

The purpose of the master thesis project is,

- To investigate and automate the drone assembly operation using Virtual Commissioning method.
- To test the performance of KUKA LBR iiwa 14 robot for force/snap fit assemblies.

The aim of this master thesis is,

- To create a virtual model of the drone assembly cell to perform Virtual Commissioning by integrating the virtual PLC loaded with a programmable logic.
- To verify and validate the behaviour of the virtual model on the physical cell.

## 1.3 Project limitations

- The thesis covers only a part of the drone production line. To include the whole production line would be a too complex task for this project scope.
- The scope of the project will be adjusted to complete within the time frame.
- The project includes only SiL virtual commissioning method and will not consider any hardware to perform the virtual commissioning process.

## 1.4 Research questions

The master thesis project aims to find results for the following research questions provided below.

- What are the methods and components required to perform Virtual Commissioning process and the best applicable area?
- Which software platforms and tools are feasible to perform Virtual Commissioning during drone assembly application?
- How will the designed automation for drone assembly cell serve the purpose of the project?
- How flexible is the designed system to changes?
- How effective is the KUKA iiwa robot in applications involving snap/force fit assemblies?

## 1.5 Report structure

The report is divided into the following chapters as documented below.

- Chapter 2, presents detailed explanation of the concepts, terms and methodologies adopted in this thesis project.
- Chapter 3, discusses the methodology and steps followed for the Virtual Commissioning process of the drone assembly cell.
- Chapter 4, provides the results of the methods adopted during the process.
- Chapter 5, contains the discussion and findings during the course of the project. The Chapter also attempts to answer the research questions.
- Chapter 6, contains the conclusions from this master thesis and presents the future work possibilities.

# 2
# Theory

This chapter provides a detailed explanation of the concepts or terms used or adopted in the thesis project. The chapter provides in-depth explanation of automation concepts, robot anatomy, KUKA iiwa robot anatomy, PLC and Virtual Commissioning.

## 2.1 Industry 4.0

The fourth industrial revolution also called as Industry 4.0 is an initiative coined by the German federal government in partnership with universities and companies in 2011 [5]. Industry 4.0 was a strategic initiative to develop advanced production systems to increase industrial production efficiency by integrating emerging technologies [5].

According to Frank [5], Industry 4.0 technologies can be divided into 2 layers: front end and base technologies as shown in figure 2.1. Front end technologies of Industry 4.0 transforms manufacturing activities such as Production, Supply Chain, Logistics and working methods using the latest technologies. Whereas back end technologies support the other with connectivity and intelligence [5].

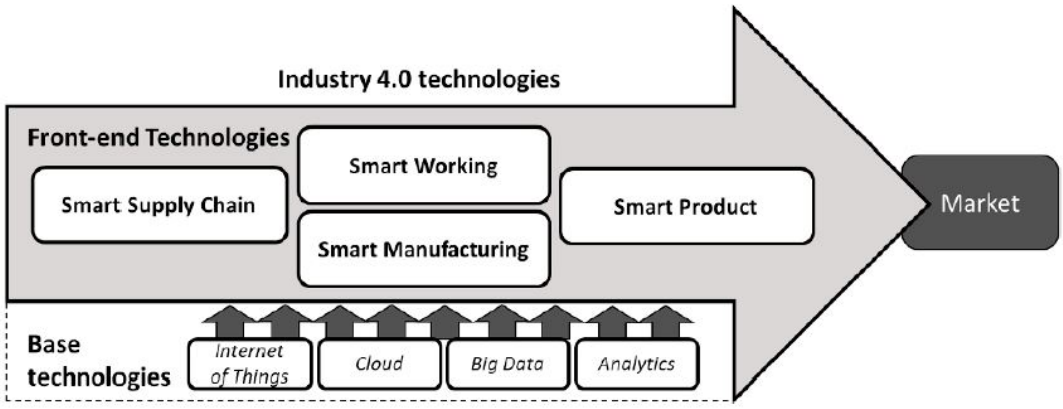



**Figure 2.1:** Theoretical framework of Industry 4.0 technologies [5]

Smart manufacturing is the beginning of Industry 4.0 and according to Frank [5], the related technologies are grouped into six different purposes.

1. Vertical integration
2. Virtualization
3. Automation
4. Traceability
5. Flexibility
6. Energy management

This master thesis includes the study of Virtual Commissioning, that is a part of smart manufacturing and is categorized under different purposes as shown in Table 2.1.

**Table 2.1:** Smart manufacturing technologies

| **Category** | **Technologies** |
|---|---|
| Vertical integration | Sensors, Actuators and PLC's |
| Virtualization | Simulation and Virtual commissioning |
| Automation | Robots |

## 2.2 Automation

Assembly systems are slightly complex because of a high degree of product variety and strategy to produce mass customized products [6]. Complex assembly systems can cause quality issues, poor ergonomics, and uncertainties [6]. So it is important to investigate how complexity can be addressed to solve these issues using the right level of automation.

A high level of automation for product realization is an important means to meet the increasing customer demands and to compete against low-cost production countries [7]. According to Mattson [8], automation is mainly used in three areas.

- To ensure precise execution of tasks
- To eliminate human repetitive and monotonous tasks to increase the stability of the output.
- To increase speed, efficiency, and security.

Automation according to Frohm is defined as *"automatic control of the manufacture of a product through several successive stages; the application of automatic control to any branch of industry or science; by extension, the use of electronic or mechanical devices to replace the human labor"* [7].

The designed automation should be adapted to the operator so that the human will perform tasks that are best suited to the human. Figure 2.2 differentiates a list of tasks that are better performed by humans and machines.

| HUMANS SURPASS MACHINES IN THE: | MACHINES SURPASS HUMANS IN THE: |
|---|---|
| ▪ Ability to detecting small amounts of visual or acoustic energy<br>▪ Ability to perceiving patterns of light or sound<br>▪ Ability to improvise and use flexible procedures<br>▪ Ability to store very large amounts of information for long periods and to recall relevant facts at the appropriate time<br>▪ Ability to reason inductively<br>▪ Ability to exercise judgment | ▪ Ability to respond quickly to control signals, and to apply great force smoothly and precisely<br>▪ Ability to perform repetitive, routine tasks<br>▪ Ability to store information briefly and then to erase it completely<br>▪ Ability to reason deductively, including computational ability<br>▪ Ability to handle highly complex operations, i.e., to do many different things at once. |

**Figure 2.2:** Task allocation [7]

### 2.2.1 Flexible automation

In common industrial applications, a customized automation process or equipment is designed to perform a specific task or to produce a specific product. These equipment are scrapped once the product's life cycle is completed. This automation approach requires more custom tools, more floor space, and more change over time to produce each new product [9]. The manufacturing industries require a more agile and flexible automation approach to face future challenges like manufacturing of low volume and high variety of products.

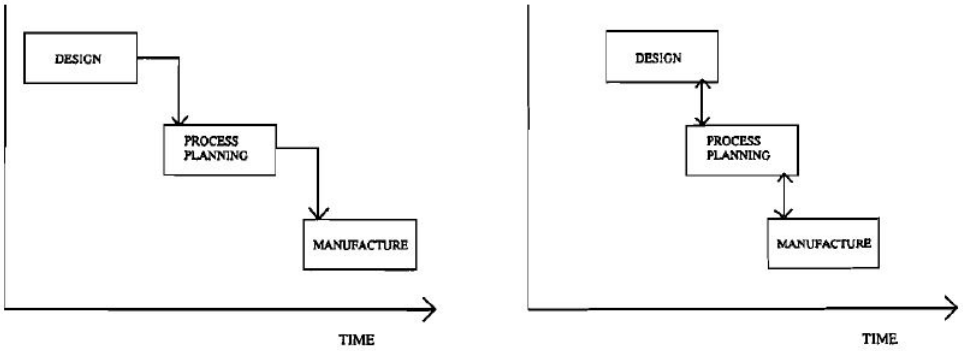
According to [9], flexible automation is defined as "the ability for a robot or system to be quickly and easily re-tasked to change product design for both low and high mix-manufacturing".

The design and integration of a flexible automation system requires three main components; sense, think, and act [9]. The sense component gathers information about surroundings through for example, vision systems, force sensors, and lasers. Think component uses the known and sensed information through PC and software to interpret, and the Act component completes the task using a robot, gantry, gripper, and actuators [9].

## 2.3 Process planning

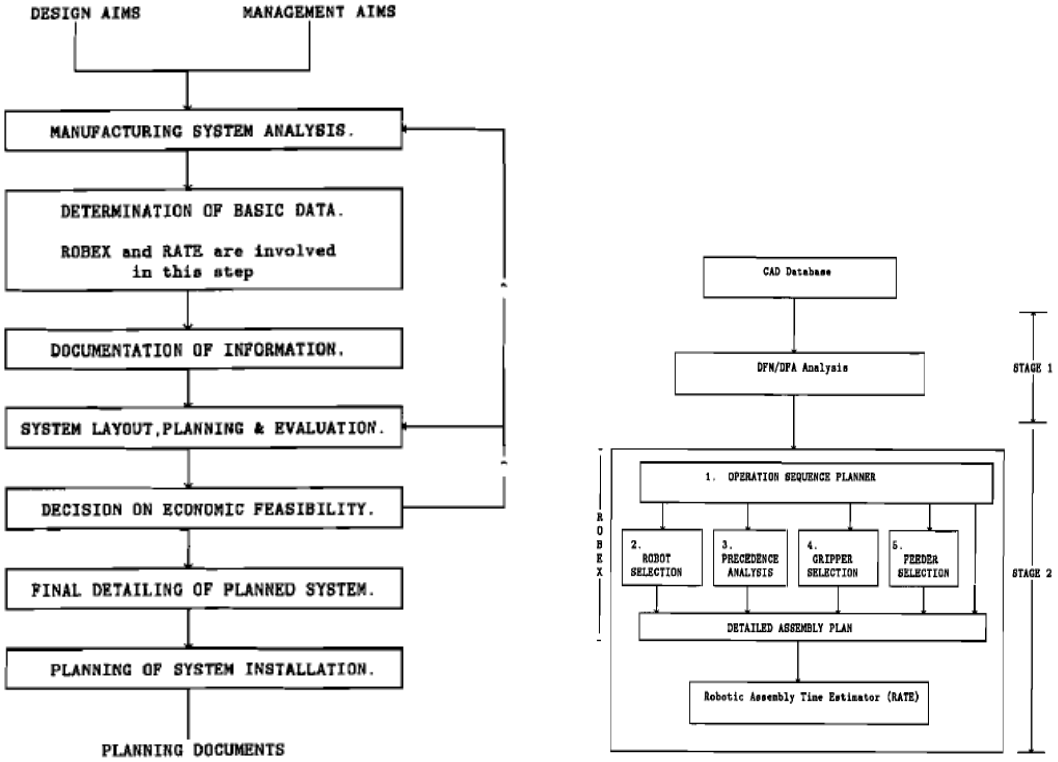
Process planning scores an important link in the manufacturing cycle. The aim is to transform raw material into finished products [10]. Browne discusses the philosophies that can be adopted for process planning in their paper [11], and this project adopted simultaneous/parallel engineering. A major requirement in this philosophy is the availability of tools, machines and technical data. The gap between design and planning cycles are reduced in simultaneous/parallel engineering whereas, traditional planning philosophy conducts design and planning cycles in a sequence [11]. Figure 2.3, represents the timeline of sequential and parallel engineering processes.

According to Browne, robotic based assembly systems are complex and presents a




**(a)** Sequential engineering **(b)** Parallel engineering

**Figure 2.3:** Planning strategies [11]




**(a)** Process planning procedure **(b)** Flow of data

**Figure 2.4:** Process planning activities and data flow [11]

methodological planning procedure using two systems, RObot Based Expert system (ROBEX) and Robotic Assembly Time Estimator (RATE) as represented in figure 2.4a [11]. The aforementioned systems are used in determination of basic data that is needed for layout planning. Figure 2.4b, represents the data flow and steps followed in ROBEX and RATE.

ROBEX is initiated with operation sequence planner, that uses a series of rules to define the sequence of the operation. Robot selection module considers technical parameters like repeatability, reachability (robot envelope), programming method etc [11]. Similar tests are also carried other methods like gripper selection, feeder selection and precedence analysis. Similar strategy is applied in the process planning stage of this master thesis.

## 2.4 Robotics

An industrial robot according to ISO 8373:2012 is an *"automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes, which can be either fixed in place or mobile for use in industrial automation applications"* [12].

- Re-programmable : Programmed motions can be changed without physical alterations.
- Multipurpose : Capable of adapting to different applications.
- Axis : Direction of motion (Rotary/linear)

### 2.4.1 Robot anatomy

According to Kihlman [13], a Robot consists of a manipulator which is divided into two sections: Body & arm and Wrist. The body & arm section is used to position the objects in the robot work volume and the robot wrist is used to orient the objects [13]. The wrist is attached to the robot arm end and an end effector is attached to the wrist. The robot manipulator consists of joints and links. Joints provide relative motion and links are the rigid members between joints.

End effectors are the special tools attached to the robot wrist to perform specific tasks. There are two types of end effectors: grippers and tools. Grippers are used to grasp and manipulate objects during the work cycle and tools are used to perform a process [13]. Tool centre point (TCP) is a working point of a tool in cartesian coordinate system and multiple TCP's can be created for a tool.

There are two types of robot motion: translational and rotary. Translational motion can be achieved through a linear or orthogonal joint, and rotary motion can be achieved through a rotational or twisting, or revolving joint [13]. Each joint has one degree of freedom and most robot possesses five or six degrees of freedom.

### 2.4.2 Robot classification

According to International Federation of Robotics [12], industrial robots can be classified based on their mechanical structure.

- **Cartesian robot:** A robot that has 3 prismatic joints and whose axes coincide with a cartesian coordinate system.
- **SCARA robot:** A robot that has 2 parallel axes to provide compliance in a plane.
- **Articulated robot:** A robot that has at least 3 rotary joints.
- **Parallel robot:** A robot that has concurrent prismatic or rotary joints.
- **Cylindrical robot:** A robot whose axes form a cylindrical coordinate system.

### 2.4.3 Coordinate systems

The coordinate system defines the position and orientation of an object in space. The coordinate systems relevant to an industrial robot are: world, robot base, user, object, flange and tool, and is shown in fig 2.5.

- **World coordinate system:** The world coordinate system is a permanently defined Cartesian coordinate system and it is the base for all other coordinate systems. By default the world coordinate system for KUKA LBR iiwa robot is located at the base of the robot [14].
- **Robot base coordinate system:** Robot base coordinate system is the Cartesian coordinate system which is located at the base of the robot and it defines the position of the robot relative to the world coordinate system [14].
- **User coordinate system:** User coordinate system is a reference coordinate system used to define motions in Cartesian space. By default the world coordinate system is used as User coordinate system, but it is also possible to define additional User coordinate system relative to world coordinate system. [14].
- **Object coordinate system:** Object coordinate system defines the position and orientation of the object relative to user coordinate system. The combination of user coordinate system and object coordinate system is used to calibrate the robot coordinates to programmed paths [15].
- **Flange coordinate system:** Flange coordinate system defines the position and orientation of the robot flange and it is always moving with the robot [14]. Flange coordinate system acts as a base for the coordinate systems which describe the tool mounted on the flange.
- **Tool coordinate system:** Tool coordinate system defines the position and orientation of the tool. The origin of the tool coordinate system is called Tool centre point (TCP). It is possible to define any number of frames for a tool and can be selected as Tool centre point (TCP).

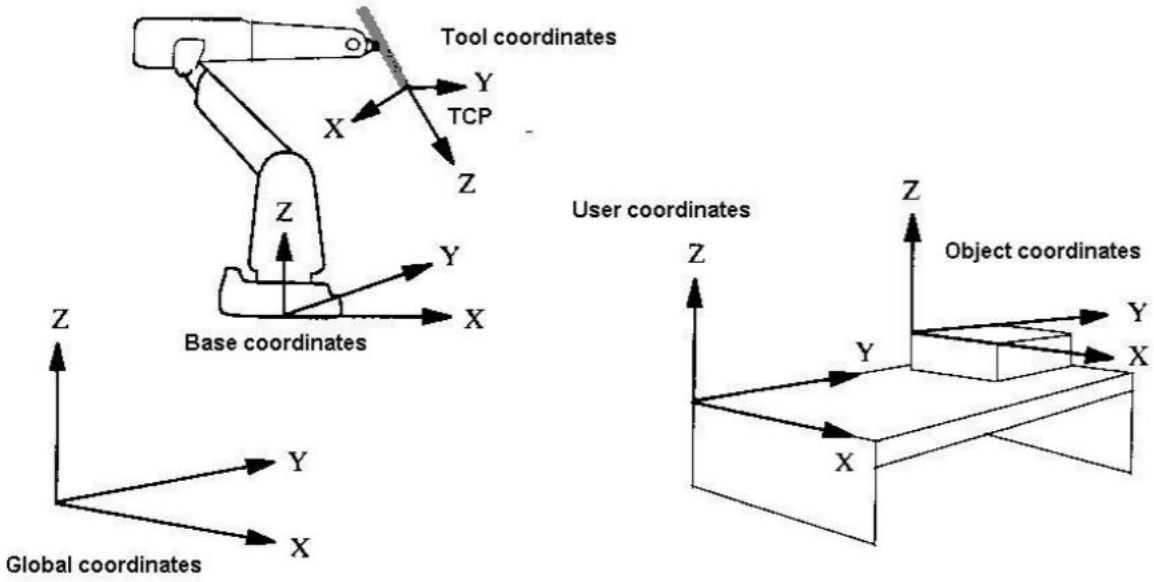



**Figure 2.5:** Robot Coordinate Systems [16]

### 2.4.4 Calibration

When calibrating an industrial robot, the previously generated offline programs from a simulation model are compared with the physical condition of the production system [17]. Robot calibration is used to identify the variations which can disrupt or stop the production system. Industrial robot actions are generally repeatable but not accurate and the accuracy errors may be systematic (deviation in robot component dimensions or their base positions) or fluctuating (thermal drift or variation in ambient temperature) [13]. The accuracy of an industrial robot depends on the robot model and brand. But using robot calibration, the accuracy of a robot can be improved by a factor of 2 to 10 [18].

The drone assembly process using the KUKA iiwa robot involves 2 types of calibration: Base calibration and Tool calibration, to eliminate any variations which can cause accuracy errors. Per suggests that, it is ideal to use terms like robot cell or robot task calibration [19]. The procedure to perform these calibration process are explained in chapter 3. Figure 2.6, represents the different calibration steps that are to be performed during robot cell calibration [19].

#### Base calibration

Several calibration methods are available to locate an object in the robot coordinate system. Zhang, has presented a quick 3 point calibration method that is used in this master thesis project [20]. The method adopts the internal encoder data of the KUKA robot. This method involves moving the robot to a reference point defining the work object. A sharp tapered tool is attached to the robot flange and is

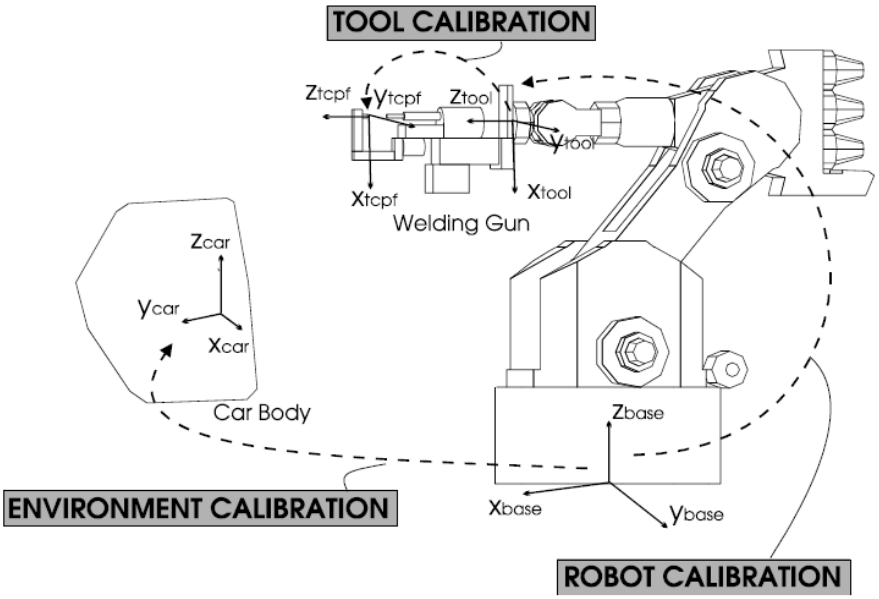



**Figure 2.6:** Robot calibration steps [19]

moved to touch another sharp tool or corner to determine the position of an object. The homogeneous transformation matrix, ${}^{\mathrm{R}}\mathrm{T}_{\mathrm{U}}$ of the base coordinate or workpiece coordinate system in defined by Zhang, in the Roll-pitch-yaw style presented in the equation 2.1 below [20].

$$ {}^{\mathrm{R}}T_{\mathrm{U}} = \begin{bmatrix} c\phi c\theta & c\phi s\theta s\psi - s\phi c\psi & c\phi s\theta c\psi + s\phi s\psi & P_{\mathrm{x}} \\ s\phi c\theta & s\phi s\theta s\psi + c\phi c\psi & s\phi s\theta c\psi - c\phi s\psi & P_{\mathrm{y}} \\ -s\theta & c\theta s\psi & c\theta c\psi & P_{\mathrm{z}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1} $$

Where, c represents cos and s represents sin. The six variables are $\mathrm{P_x}$, $\mathrm{P_y}$, $\mathrm{P_z}$, $\phi, \theta, \psi$

### Tool calibration

Specific tools like grippers or weld guns are mounted on the robots to perform desired operations [19]. The position and orientation of the cartesian space becomes important along with the exact position of the tool. This is achieved by the tool calibration process. In order to locate the toolframe origin, the robot needs to rotate around it [19]. A pointer tool is placed in the environment. With the flange system activated, the tip of the pointer tool is positioned as close as possible to the attached tool using at least four different orientations. Thus X, Y and Z coordinates of the toolframe origin is obtained. The direction of the toolframe axes is further determined by moving the robot along directions of two toolframe axes [19]. The detailed methodology of the tool calibration process is presented in section 3.5.1.

Calibration process is an operator-dependent method, the results from this method are individual and can vary for each individual person. The manual calibration

method is not efficient when high accuracy is required. However, this may still be used as a suitable calibration method when high robot accuracy is not required, and if there is a cost constraint [15].

### 2.4.5 Robot programming

Programming time and quality are important factors to make the robot work efficiently. The time and quality of the program depends on the robot programming methods and their supporting systems. According to bolmsjö [21], the robot programming methods are classified based on where the programming takes place and how it is done. Programming can either be performed online or offline and how the program is done depends on the abstraction level defined [21].

#### Online programming

Online programming is a robot programming method where the robot system is used to program the robot. Online programming is frequently used for simple and less complex tasks, but it is hard to follow and maintain during complex tasks that require frequent reprogramming or setup [21]. One major disadvantage of using an online programming method is the programmer occupies the robot while programming and this makes the robot system or the production system to which that robot is connected to stop [21]. Thus robot programming time plays an important role in improving production efficiency.

#### Offline programming

Offline programming is another robot programming method where external software is used to program the robot. This method requires pre-knowledge about computers and computer programming languages to program the robot. Offline programming is done while the robot is in operation and this saves a significant amount of production time while performing complex tasks. Offline programming needs to be checked for its correctness against geometric tolerances during deployment and sometimes it might require some final touch up and checks to make it correct [21]. Even though the deployment process takes some time to correct the robot program, it is much shorter compared to the online programming method.

Offline programs can be much more complex than online programs due to added instruction sets, data types, and decisions, this results in small deviations in the robot program when deployed to a real robot. Modern programming platforms provide necessary supports to compensate these issues by providing specific macros for certain sub-tasks, optimizing paths for a specific purpose, assist in writing and editing, simulation, and checking program [21].

### 2.4.6 KUKA LBR iiwa

KUKA LBR iiwa is a lightweight, sensitive and human robot collaboration compatible robot. The basic components of the robot includes: manipulator, controller,

smartPAD and connecting cables [22]. An overview of the robot system is shown in figure 2.7.

LBR iiwa robot comes in two variants with different payload capacities: 7 kilograms and 14 kilograms. Robots with an integrated sensor system are fundamentally changing the production processes of the future and laying foundations for innovative and sustainable production systems. This enables humans and robots to cooperate and work together on sensitive tasks. LBR iiwa robot is equipped with position sensor, torque sensor and temperature sensor in all its 7 axes and all joints can be programmed individually [22]. This enables the automation of delicate assembly operations with force-controlled joining operations and process monitoring, and thereby avoiding costly rejections and collisions.


**Figure 2.7:** Overview of robot components [23]

### Position and Orientation

The position of an object in Cartesian space is defined by specifying translation and rotation relative to reference coordinate system [14]. Six coordinates are used to define the position of an object in Cartesian space and are shown in table 2.2. The basic data and axis data of KUKA LBR iiwa robot is shown in table 2.3a and 2.3b.

**Table 2.2:** Position and orientation [14]

| Coordinate | Description |
|---|---|
| **Translation** | |
| Distance X | Translation along the X axis of the reference system |
| Distance Y | Translation along the Y axis of the reference system |
| Distance Z | Translation along the Z axis of the reference system |
| **Rotation** | |
| Angle A | Rotation about the Z axis of the reference system |
| Angle B | Rotation about the Y axis of the reference system |
| Angle C | Rotation about the X axis of the reference system |

### 2.4.7 Force and Toque control

Compliance control in a robot is required for robot applications like grinding, assembling, polishing and human-robot interaction, which requires both position and torque control [24]. Compliance in an industrial robot indicates flexibility and suppleness, on the other hand, non-compliant robots are rigid and have predetermined positions and trajectories. Even though most robots require rigidity to achieve the necessary precision, a rigid robot cannot adapt its motions in case of disturbances or unplanned situations [25]. Due to increased complexity in assembly lines, robots are required to feel if anything goes wrong and this is why force-torque sensors are necessary for the robotic industry [25].

A force-torque sensor in a robot detects the different forces/torques acting on the robot joints, wrist, or tool in 3 geometric axes [25]. The force-torque sensor provides a sense of feeling to the robot to sense its surrounding. Torque controlled robots are essential for safe human-robot interaction and to support smooth detection of delicate external events. For this purpose, good control to hide inertial forces is required [26]. KUKA iiwa robot is equipped with position and joint torque sensors in all its 7 axes to measure and react to external forces and torques [14] and it is also possible to measure force/torque on a specific axis for a specific application. This feature enables the automation of delicate assembly tasks for force-controlled joining operations and process monitoring [22].

The F/T sensors of the KUKA iiwa robot are used to measure external torques. The sensor is capable of measuring both force and torque acting along the 3 axes, and they are defined by the components Fx-Fy-Fz and Mx-My-Mz. Bélanger suggests that, there are various ways to measure force and torque values, generally most sensor manufacturers use strain gauges with a specific orientation [25]. However, there are some sensors that use gauges that digitize the measurements from the start [25]. The force-torque sensor produces output as either analog or digital signal, and the signal is the communication method used to input the robot through a field bus [25]. Analog signals are not ideal for industrial robotic applications due to their high level of electromagnetic noise [25]. Most robot manufacturers also provide software packages to use the information received from sensors and allow the user to program the

**Table 2.3:** Basic data and Axis data for LBR iiwa 14 R820 [23]

**(a)** Basic data

| | **LBR iiwa 14 R820** |
|---|---|
| Number of axes | 7 |
| Number of controlled axes | 7 |
| Volume of working envelope | 1.8 $m^3$ |
| Pose repeatability (ISO 9283) | ± 0.15 mm |
| Weight approx. | 29.9 kg |
| Rated payload | 14 kg |
| Maximum reach | 820 mm |
| Protection rating | IP 54 |
| Protection rating, in-line wrist | IP 54 |
| Sound level | < 75 dB (A) |
| Mounting position | Floor |
| Controller | KUKA Sunrise Cabinet |

**(b)** Axis data

| | **Range of motion** |
|---|---|
| A1 | ±170 ° |
| A2 | ±120 ° |
| A3 | ±170 ° |
| A4 | ±120 ° |
| A5 | ±170 ° |
| A6 | ±120 ° |
| A7 | ±175 ° |
| | **Speed with rated payload** |
| A1 | 85 °/s |
| A2 | 85 °/s |
| A3 | 100 °/s |
| A4 | 75 °/s |
| A5 | 130 °/s |
| A6 | 135 °/s |
| A7 | 135 °/s |

robot with high-level commands.

The different terms used for force and torque analysis of the KUKA iiwa robot as explained in [14] are listed below.

- **External axis torques:** The external axis torques are generated from the forces and torques occurring due to robot interaction with its environment. External axis torques are not directly measured but are calculated using the dynamic robot model. The accuracy of these torques depends on the dynamics of the robot motion and the interactive forces of the robot with its environment.
- **Internal axis torque:** These are the measured axis torque of a robot.
- **Cartesian Forces:** These are the forces acting along the X, Y, and Z axes of a defined frame (Ex: TCP).
- **Cartesian Torque:** These are the torques acting about the X, Y, and Z axes of a defined frame (Ex: TCP).

Even with all the advantages of having compliance control in the robot, the robot torque controller is not consistent with its performance and shows variations in their results, Vinay [26] mentions some of the factors which could affect the performance of the torque controller and are listed below.

- Transmission nonlinearity in the motor to joint gearing (as shown in figure 2.8) injects vibrations and limits the performance of the torque controller. A fluctuating gear reduction between the motor and joint could be a reason for this nonlinearity which causes periodic deflection between joint and motor positions.

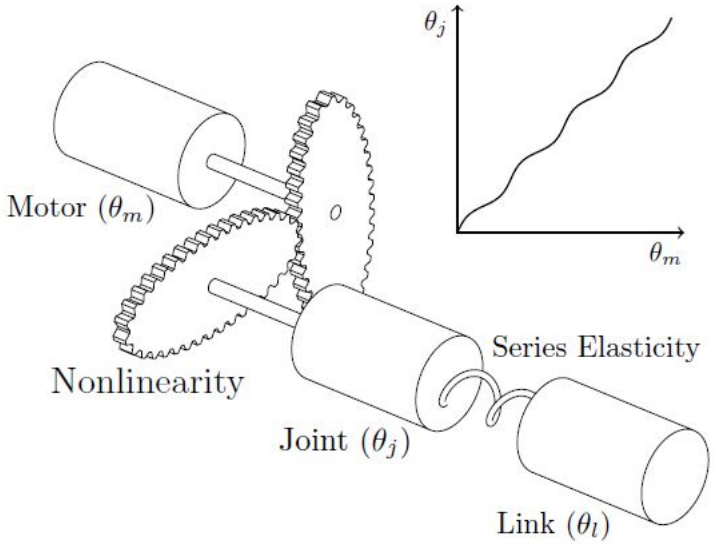



**Figure 2.8:** Transmission nonlinearity [26]

- External torque estimation shows large velocity-dependent fluctuations and velocities themselves show significant vibrations.

- KUKA provides Fast Robot interface (FRI) which allows real-time control of robot up to 1 kHz control loop rates. The FRI accepts commands for the motor position or joint torque and these torque commands are passed through a feasibility filter which imposes a maximum slew rate and torque control is suppressed when starting from zero velocity at torque levels below 1Nm.

## 2.5 Programmable logic controller (PLC)

Today's automation requires machines and equipment to run automatically and want machines to do everything. Earlier the industrial equipment was operated manually, later through traditional relays, and now using programmable relays PLC takes the automation to its highest level.

According to [27], PLC is a special form of microprocessor-based controller that uses the programming instructions to perform functions like sequencing, timing, counting and arithmetic to control machines and processes.

PLC's are designed uniquely to control the industrial equipment under harsh conditions and they do not use computers to control the equipment due to their instability and prone to crashes and power surges [28]. PLC's are designed not to fail under harsh conditions and even if they failed they are made to fail in the safest way possible [28].

### 2.5.1 PLC Configuration

The basic components of the PLC system are CPU, power supply unit, programming device, and memory unit. The schematic representation of the PLC system is shown in figure 2.9. The CPU consists of a microprocessor which interprets the input signal based on the program memory to perform control action and then send these actions as output signals [28]. The power supply unit's function is to convert the main AC voltage into low DC voltage as required by the controlling devices[28]. The memory unit stores the program containing control actions to be performed by the microprocessor and also the data from the input and output signals are stored [28].

### 2.5.2 PLC Programming

IEC 61131 defines the standards for PLC and IEC 61131-3 defines the PLC programming languages [28]. IEC 61131-3 standard provides five PLC programming languages [28].

1. Ladder diagrams (LAD)
2. Instruction list (IL)
3. Sequential function chart (SFC)
4. Structured text (ST)
5. Function block diagram (FBD)

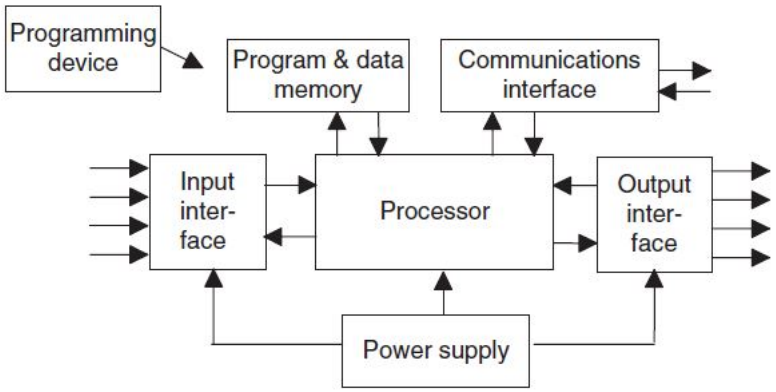



**Figure 2.9:** PLC System [28]

This master thesis follows Ladder diagrams (LAD) programming language to write the PLC program.

PLC's can be programmed in handheld devices, desktop consoles, and personal computers. PLC manufactures have programming platforms to program their PLCs like RSLogix for Allen-Bradley, TIA portal V15.1 for Siemens, MELSOFT for Mitsubishi, CX-One for OMRON, and ProWorx 32 for Telemecanique [28]. TIA portal V15.1 platform has been used in this master thesis to program Siemens PLC.

TIA portal V15.1 is a standard software package developed by Siemens to configure and program Siemens PLC's. The software supports all programming languages defined by IEC 61131-3 [28].

## 2.6 Virtual commissioning

Real commissioning of a production system involves a real plant and real controller whereas, Virtual Commissioning is an industry 4.0 technology that uses a virtual model and real controller to enable full emulation of production systems for verification [29]. A graphical comparison of real and Virtual Commissioning is shown in figure 2.10. A virtual model is an imitation of physical and logical behaviors of a real production system and will work exactly like the real station.

### 2.6.1 Virtual commissioning process

The procedure to perform Virtual Commissioning (VCom) process involves the following steps and the schematic representation of this procedure is shown in figure 2.11.

1. Data collection
    - Literature research
    - Technical data

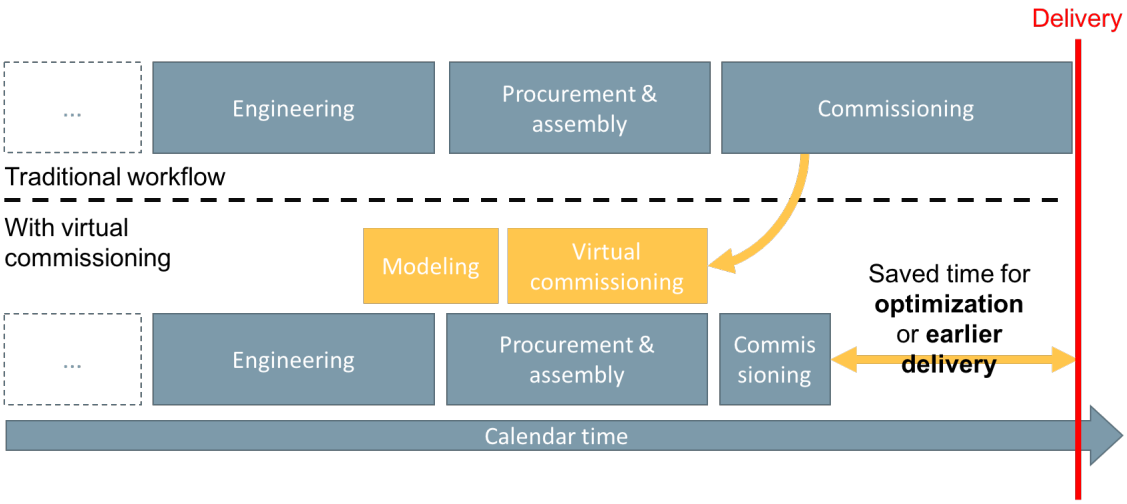



**Figure 2.10:** Virtual commissioning work flow [30]

- Software selection

2. Process planning
3. 3D Modelling
4. Simulation
   - Visual Components
   - Tecnomatix Process Simulate
5. Physical implementation and verification
   - Robot calibration
   - Robot programming
   - Force/torque analysis
6. Logical control programming
   - PLC programming
7. Virtual commissioning
   - SiL Method

The process is initiated with the data collection stage, that involves literature research, collection of technical data and selection of software for modelling and simulation. The next stage is process planning, that involves, preparation and decision making of robot placement, assembly operation sequence. This stage is succeeded by 3D modelling. In this stage, CAD model design and assembly workbenches are used in preparation of components. This thesis required assembly of schunk gripper components to be used in the simulation environment. Simulation stage involved in setting kinematics, layout preparation and simulation of assembly process. Physical implementation and verification stage involves setup of the robot cell through robot calibration and programming. Logical control programming involves PLC programming that is loaded on to the virtual controller and prepared for the final stage, i.e. Virtual commissioning. The processes represented in figure 2.11 are iterative, as modifications/improvements are performed.

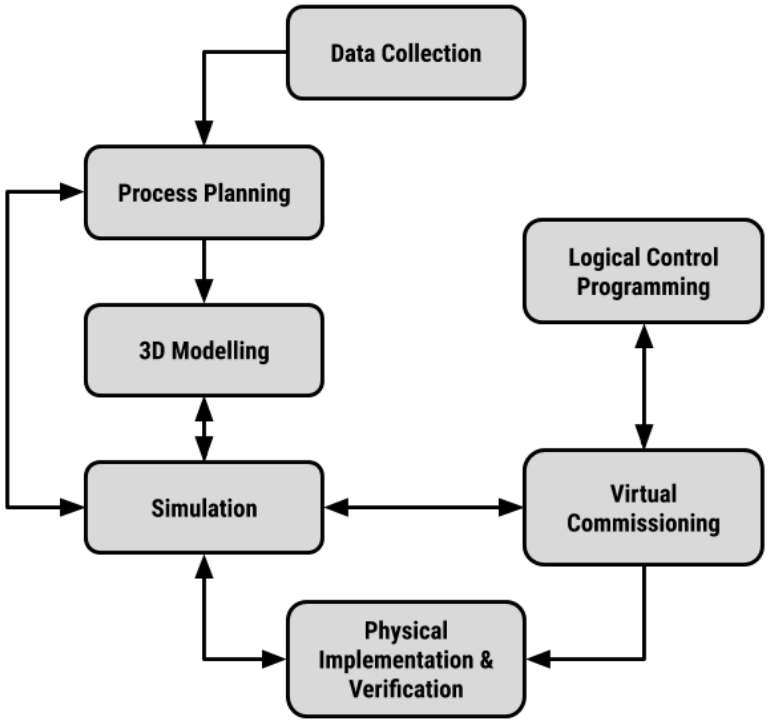



**Figure 2.11:** Virtual commissioning process

### 2.6.2 Virtual commissioning methods

With all the latest technologies, it is now possible to test everything in software; the virtual model with virtual PLC can be simulated using different software packages. On the other hand, it is also possible to simulate a virtual model using real PLC. Based on these principles Virtual Commissioning methods can be distinguished as Software in loop, Hardware in loop and Hybrid simulation [32].

- **Software in loop (SiL):** All components used in this method are a digital representation of the real station. The virtual model is controlled by a virtual PLC as shown in fig 2.12. This method is relatively cheap compared to other methods and also fast to deploy. Since this method doesn't include any real equipment, several safety-related issues can be avoided.
- **Hardware in loop (HiL):** This method tries to use maximum number of hardware components as possible along with virtual model to commission a production system as shown in figure 2.12. The setup created using this method is more identical to the real system and the PLC code is tested using a real controller. Due to these facts, the implementation cost of this method is relatively high. The involvement of real industrial equipment can result in several safety-related problems. An additional downside to this method is low flexibility when changes occur.
- **Hybrid simulation:** This method uses the 3D model with a virtual robot controller (developed by robot manufacturers)[32]. In this way, it is possible to run the code on the real PLC with the exact hardware configurations that

are going to be used [32]. This method is more flexible and any changes made to the plant design can be adapted into the 3D model.

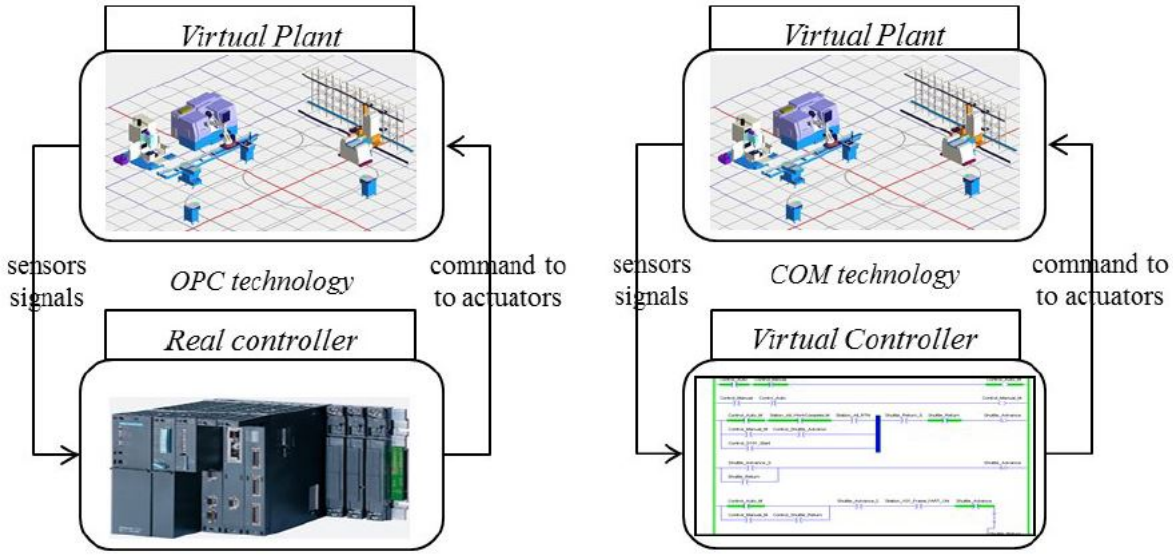



**Figure 2.12:** Virtual commissioning methods [1]

### 2.6.3 Benefits of virtual commissioning

Some of the advantages of using Virtual Commissioning before implementation of production systems are,

- Since, Virtual Commissioning is performed before physical implementation of the station, deviations from the requirements can be fixed at an earlier stage which results in decreased start-up time of the plant. According to Mortensen [29], Virtual Commissioning can lower the commissioning time by up to 63%.
- Virtual Commissioning allows us to perform programming and debugging simultaneously which results in reduced debugging and correction efforts during real commissioning [1].
- Using Virtual Commissioning, companies will have the ability to test more complex scenarios with robots and mechatronic systems [32].
- Virtual Commissioning reduces the product's time to market which provides faster feedback to recognize probable issues. [32].
- Since, possible errors are fixed at the Virtual Commissioning stage, the quality of the real commissioning is increased.[32].
- Virtual Commissioning increases the predictability of process lead times which makes the lead-time estimation accurate [33].
- Shorter lead time and increased predictability makes the system more flexible and helps to make decisions faster [33].
- Several safety-related issues can be avoided using virtual commissioning method.

### 2.6.4 Drawbacks of virtual commissioning

The success of Virtual Commissioning project depends on many factors and some of the current problems with implementing Virtual Commissioning in industries are

- Lack of competences and integration experience. There might also be resistance among investors to start the Virtual Commissioning project because of this.
- The creation of a virtual model requires significant time and effort since most of the parts are need to be modeled individually and also due to lack of access to CAD libraries.
- The success of Virtual Commissioning implementation depends on the quality of the virtual model. Failure to do so will result in inaccurate commissioning. Also, it is important to update the virtual models according to specification of the plant to avoid gaps between the real plant and virtual model.

# 3
# Methodology

This chapter presents the methodology adopted for conducting Virtual Commissioning of the drone assembly cell. The stages involved in this process are presented in section 2.6.1 of chapter 2. These stages were devised from literature research of various technical papers on Virtual Commissioning and the same are presented in the below sections.

## 3.1 Data collection

This stage involved collection of data required to perform the project such as literature research, software selection etc. The input from this stage primarily aided in establishing and execution of other stages involved. The methods followed are explained in this section

### 3.1.1 Literature research

Data collection included a literature research on the topics related to theory of Virtual Commissioning, methods and stages followed in the process. This stage also included research on simulation software available along with PLC programming methods and possibilities of mapping the signals with the virtual model. Technical papers with the aforementioned topics were found through Chalmers library website and google scholar. Keywords such as Virtual Commissioning, KUKA iiwa, Tecnomatix Process Simulate, Visual Components, KUKA Sunrise Workbench, Force/-Torque control, TIA Portal, and PLCSIM were used to find technical papers relevant to the thesis project topic. Data on robot anatomy of the KUKA iiwa robot, force/-torque controls and topics relating to industry 4.0 where, Virtual Commissioning could play a vital role were researched with aid of technical papers and manuals.

### 3.1.2 Technical data

In addition to literature research, the drone factory layout information including conveyor, pallet and station dimensions were collected physically at SII-Lab, Lindholmen. CAD models of the drone frame, drone motor, fixture setup for assembly and Schunk gripper were collected from personnel in-charge at SII-Lab. Standard components such as assembly table, assembly rails were downloaded from online CAD libraries. The access to Visual Components libraries were used for conveyors and KUKA iiwa robot.

### 3.1.3 Software selection

This stage included the process of shortlisting and finalizing simulation software available in the market that cater to the project requirements. Various simulation software available in the market were researched for features and examined for cross platform compatibility. A major requirement for the project was the access to internal libraries that could improve efficiency of simulation process.

In addition to the online research of software applications, a brief interview of our thesis supervisor was conducted. The interview was aimed at understanding virtual commissioning possibilities in the Swedish automotive industry. It was also possible to get suggestions on software for Virtual Commissioning used at automotive industries like Volvo cars and information on software available with Chalmers university. Similarly, modeling software required for the project to access CAD files and assembly were selected.

## 3.2 Process planning

This stage involved planning of the drone assembly process. Technical data collected, were used in the analysis. The sequence of assembly operation was determined using planning activities from figure 2.4. This included activities like placement of the KUKA robot, fixture location, drone component placement on pallet etc. Ideal placement of the robot deemed necessary to prevent collisions and reach test to confirm reachability of the robot during assembly and while picking components from pallet. Section 2.3 in chapter 2 presents the planning process activities and philosophy that are generally adopted in these projects. The results obtained from this stage are discussed in the further chapters. The process was iterated to test several alternatives to the assembly process that assisted in proposing the ideal solution.

## 3.3 3D Modelling

3D modelling stage primarily involved development and assembly of CAD parts to be used in simulation platforms. CAD Part files of the Schunk Co-act gripper were downloaded to Catia V5 and assembled using constraints on the assembly workbench. The CAD product was further exported with .stp file format to be used in simulation applications. CAD model of the gripper fingers were procured from the official Schunk catalogue and 3D printed, to be installed on the real gripper. CAD parts of drone components, i.e. drone frame, motors and assembly fixture were obtained from SII-Lab personnel and exported to simulation software for assembly process simulation. Access to online library on Visual Components, provided us with a possibility of exporting standard components such as the KUKA iiwa robot, conveyors, table along with assembly fixture in .jt format to be used on Tecnomatix Process Simulate as well. Several iterative improvements were made to the fixture in this stage that aided in improving the assembly process and reduce cycle time.

## 3.4 Simulation

In this stage, with the use of simulation platforms, a layout with robot and assembly fixtures were prepared. The layout also included the conveyor system as established at SII-Lab. The process of assembling drone frame with drone motors using snap/-force fitting was created with the KUKA iiwa robot. Simulation of the process was performed with two software applications discussed in sections below.

### 3.4.1 Visual components

#### Model a component

The process for modelling the Schunk gripper is explained in this section. Since, Schunk Co-act gripper that was installed on the KUKA iiwa robot at SII-Lab was not available on Visual Components library, CAD product was imported and modelled on the simulation platform. Similar process was followed for modelling the KUKA robot positioner. The CAD model of the Schunk gripper parts that include gripper fingers, adapters, connection hubs were assembled in CATIA V5 on assembly design workbench and imported. The procedure for modelling the gripper is explained in Appendix A, section A.1.

#### Layout preparation

The layout of the Drone factory as established at SII-Lab was created. The layout included only the KUKA iiwa robot station where assembly of the drone frame with the drone motors was planned. The layout was created by drag and drop option available on Visual Components. Conveyors available in the library were dragged onto the main window and dimensions were altered as available at SII-Lab. The conveyors were attached using the PnP option available on Visual Components.

A robot station consisting of an assembly table and fixture was created through the drag and drop option. KUKA iiwa 14kg model that was used in the project was also available in the online library under robots category. The robot was dragged onto the main window. However, CAD model of the drone assembly fixture was imported, as it was not a standard component. Further, robot positioner and Schunk gripper used in SII-Lab was not readily available in the library. CAD models were imported to Visual Components and modelled to activate the PnP option as explained previously.

#### Assembly process simulation

The robots were programmed after modeling the gripper to simulate the assembly process of drone frame with motors. The process involved flow of drone components on a 250 $\times$ 250mm pallet through the conveyors, that are placed by a Motoman robot. The assembly was carried out with the KUKA iiwa robot. The KUKA iiwa robot was attached with the modeled Schunk gripper using PnP option. Program tab in Visual Components was used to program the robots. The sequence was created in the program editor tab by selecting the robot. Snap function was used to choose the

pick location. Further, Linear (LIN) or Point to Point (PTP) function were chosen as robot movements to the pick location. Signals to start or stop the conveyors or grasp or release of gripper were called at required program steps. Similar steps were carried out to complete the drone assembly process. The procedure for programming robot for assembly process is explained in Appendix A, section A.2.

### 3.4.2 Tecnomatix process simulate

#### Setting kinematics

Since there was no library access on standalone Tecnomatix Process Simulate, components such as gripper, KUKA robot, conveyors along with the fixtures etc. were exported with .jt file format from Visual Components and imported on Tecnomatix Process Simulate with .cojt format. The KUKA iiwa robot was imported from Visual Components, and kinematics was set for the same. The process involved, setting the number of joints, links, joint motions, joint movement limits, baseframe coordinate origin and flange coordinate origin for the robot. Similar process was also followed for the schunk gripper, the base coordinate origin and Tool centre point (TCP) were set. The joint type and translational distance for gripper fingers were set in the kinematics editor. The detailed procedure for setting kinematics of KUKA iiwa robot followed on Tecnomatix Process Simulate is explained in Appendix B, section B.1.

#### Layout preparation

The components that are required to build the layout including robot, gripper, conveyors, drone parts etc were imported to Tecnomatix Process Simulate and arranged according to the layout available at the SII-Lab. The parts were moved using the manipulate option available on the main window and dragged along the axes or moved from one axis to another. The schunk gripper was mounted on the robot using mount tool option in the robot tab. Pose editor option was chosen to set the release and grasp positions for the schunk gripper and home position for the KUKA robot. The layout model created in this step is presented in the next chapter. Frames were created according to coordinates obtained from calibration procedures and Robot, conveyors and other parts were arranged accordingly to match the layout accurately as in SII-Lab.

#### Assembly process simulation

The assembly process of drone components was performed by assembling drone motors with drone frame. The processes for the assembly were created in the process tab of Tecnomatix Process Simulate. The process was designed in a way that the components are transported through pallets. The pallet stops at the sensor placed on the conveyor. Object flow operation was chosen by providing start and end coordinate points for drone components. Robot operations were prepared using generic robotic operation. This feature provided coordinate points for the robot movement. Gripper operations of grasp and release were performed using new gripper operation

function. The procedure for simulation is explained in Appendix B section B.2.

## 3.5 Physical implementation and verification

The stage involved setting up the physical components of drone assembly station. The robot used in the project was calibrated and fixture was assembled onto the station. Robot programming of KUKA iiwa was performed in this stage simultaneously with calibration and then synchronized. Additionally, Force/torque analysis with the KUKA robot was conducted. The process is explained in this section.

### 3.5.1 Robot calibration

The KUKA iiwa robot available at SII-Lab was positioned at the allocated station physically. The controller and sunrise workbench application were started. A new sunrise project was created as shown in figure 3.1 and the robot program was written in a java source file. The file path was `/Sii_Lab_LBR_iiwa_01/src/application/KUKAIMSX30.java` and is displayed under the package explorer window in the left corner as shown in figure 3.2. The application data window at the right corner of the window as shown in figure 3.2 consists of frame information, where, coordinates (via points) were added with respect to the base coordinates that were obtained from base calibration. Similarly template data tab as shown in figure 3.3 consists of templates for tools and workpieces. Tool data for schunk gripper; and pointer tool used for base calibration, were added here along with TCP frames. The values for TCP were obtained through tool calibration and synchronised with sunrise workbench.

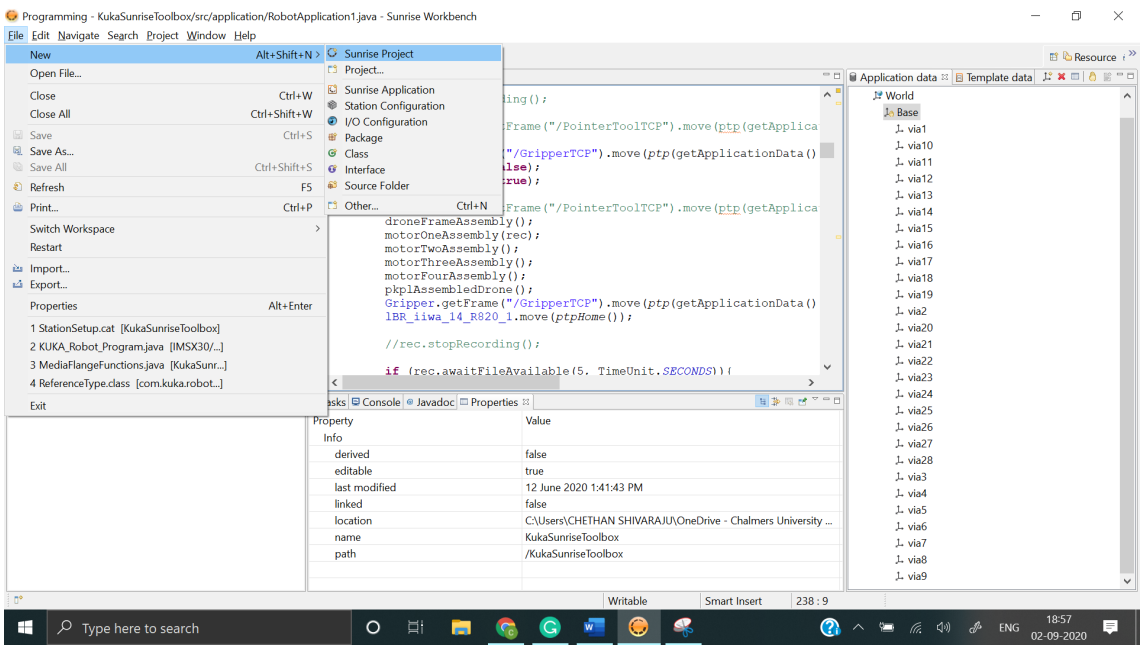



**Figure 3.1:** Start sunrise application

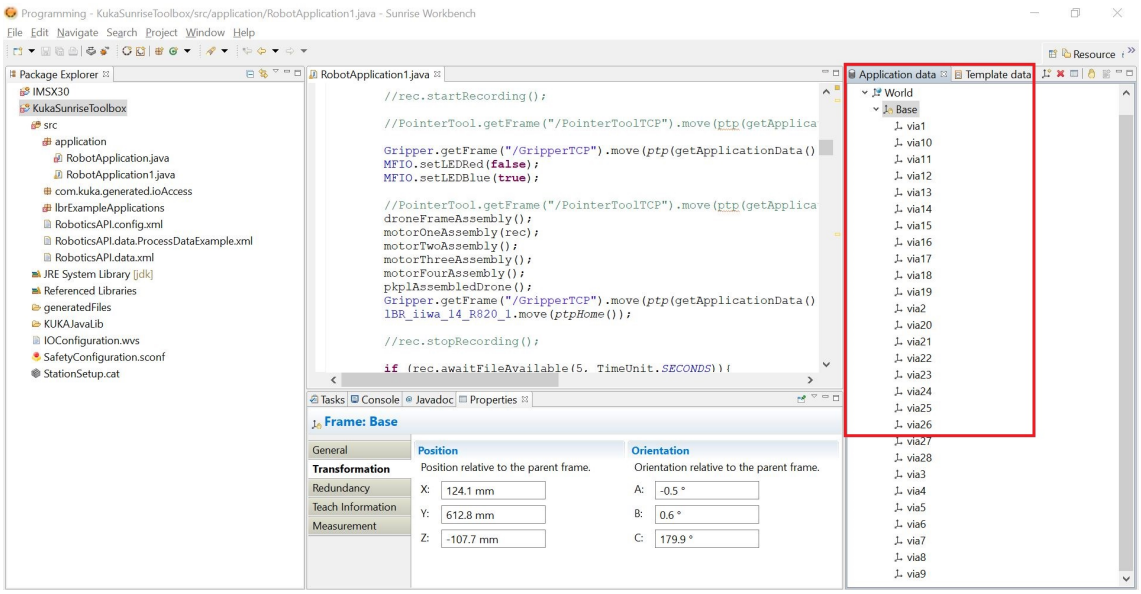



**Figure 3.2:** Sunrise application data

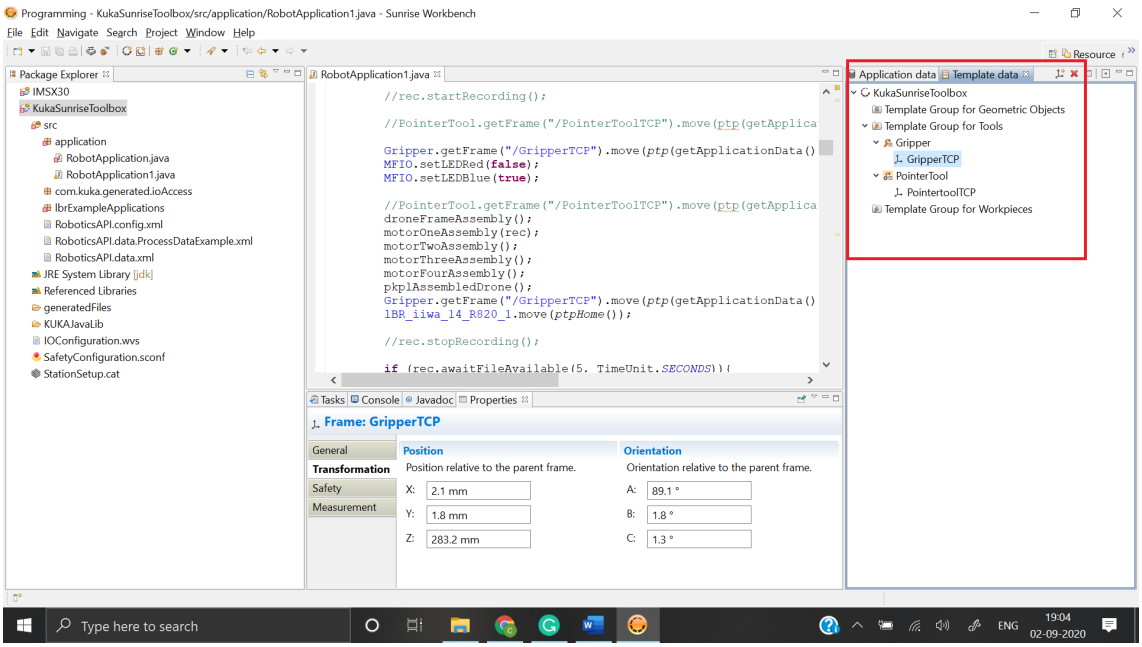
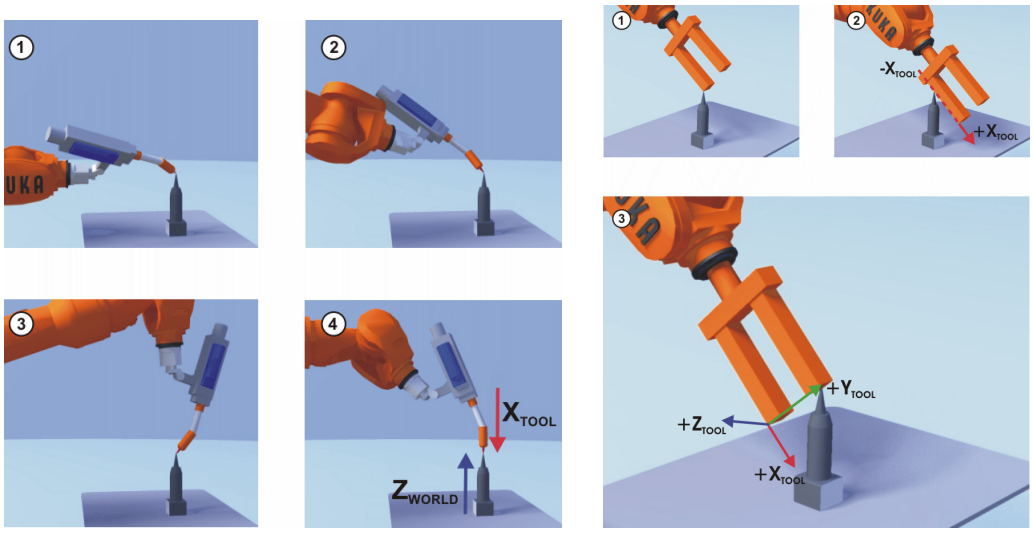



**Figure 3.3:** Sunrise template data

The robot calibration procedure includes calibration of the base and tool coordinates that are performed using the manual calibration method as mentioned in section 2.4.4.

### Tool calibration

Tool calibration is a process of assigning a Cartesian coordinate system i.e. tool coordinate system to a tool mounted on the robot mounting flange [14]. The tool coordinates i.e. Tool centre point (TCP) was defined with the calibration procedure and synchronized to sunrise workbench. The defined point or TCP will interact with objects in the environment. Consequently, not only the flange coordinate frame is important but also the location of interacting parts with respect to the flange coordinate frame. Tool calibration was performed for the schunk gripper and a pointer tool. The pointer tool was used in base calibration procedure. The procedure for tool calibration as followed consisted of the following 2 steps, XYZ 4-point method and ABC 2 point method.




**(a)** XYZ 4-point method [14] **(b)** ABC 2-point method [14]

**Figure 3.4:** Tool calibration methods

- **XYZ 4-point method :** This method was used to define the origin of the tool coordinate system. The TCP of the tool to be calibrated was moved to a reference point from different directions [14]. A pointer was placed at a corner of the table and chosen as the reference point for the procedure. Figure 3.4a refers to the possible movement of the tool in four different directions. Step by step procedure is explained in Appendix C section C.1.1.

- **ABC 2-point method :** This method was used to define the orientation of the tool coordinate system. The method is used for tools with edges and corners like schunk gripper that can be used for orientation purposes by defining

a point on the X axis and another on the XY plane. This method is necessary to define the axis directions with precision [14]. Figure 3.4b refers to the possible movement of the tool during the procedure. Step by step procedure is explained in Appendix C section C.1.2.

### Base calibration : 3-point method

The base calibration process assigns a cartesian coordinate system i.e. base coordinate system to a frame selected as base [14]. 3-point method was followed as the procedure, provided in the KUKA sunrise manual. The origin of base was defined at a user defined point i.e. edge of the table with fixture. Further, positive direction of the x axis and a point on the xy plane were recorded to obtain the base coordinate system. Pointer tool was used in the procedure to define the points accurately. The advantages of base calibration led to jogging of TCP along the edges of the work surface of workpiece. Additionally, via points were taught relative to the base [14]. The procedure also aided in calibrating the simulation environment for increased accuracy of coordinate points. Figure 3.5 refers to the possible movement of the tool during the procedure. Step by step procedure is explained in Appendix C section C.2.1. The procedure was repeated by moving the robot to a different position and record the base coordinates.

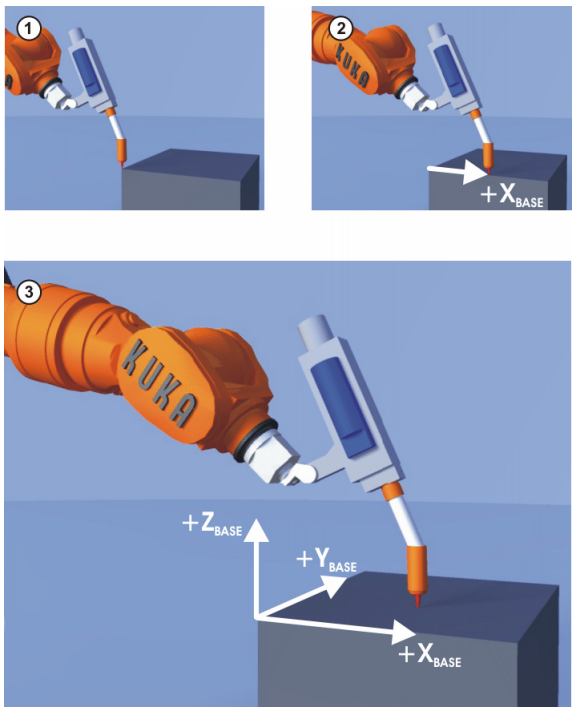



**Figure 3.5:** 3-point method [14]

### Hand-guiding method

As mentioned previously, the KUKA iiwa robot can be used extensively in human robot collaboration. The handguiding feature of the robot offers extensive benefits

during motion guidance and programming. This feature was employed in conducting experiments to test the robustness and accuracy of the robot. The experiment conducted aided in locating the origin of the base coordinate system post 3-point calibration methods explained above and demonstrate that the hand-guiding feature can be used as an alternative to base calibration process.

The experimental setup consisted of a 3D printed holder with a tool adaptor hub, that was placed at a defined distance from the table. The tool mounted on the robot was detached. The handguiding device was activated and moved towards the tool adapter hub for assembly. The robot was stopped at this position and cartesian coordinates of the flange were recorded. Flange coordinate position with respect to base coordinates provided the deviation of the flange coordinate origin from the base origin. Since, the base origin on the simulation platform was set at the bottom left corner of the table, this method aided in realizing the accurate position and the necessary deviation of the base origin to be maintained on simulation platform for increased accuracy. Figure 3.6 shows the procedure followed for the method.

#### Variation analysis

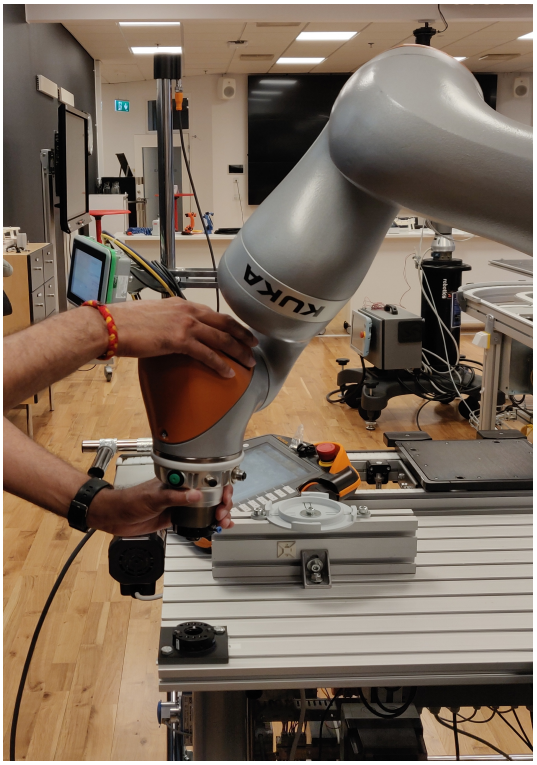
An additional experiment was conducted on the KUKA iiwa robot to determine the variation in the base coordinates from the 3-point and hand-guiding calibration method.

The robot was positioned near the station and base calibration was conducted using both 3-point and hand-guiding method. The procedure for the methods are explained in the above sections. The results from the method was tabulated as 3-point and hand-guiding calibration values. The flange coordinate position with respect to world coordinates are noted during the hand-guiding method (robot base coordinate system is identical to the world coordinate system by default [14]). The robot position was changed and the base calibration procedure was repeated. The results from the experiment is presented in the following chapter.

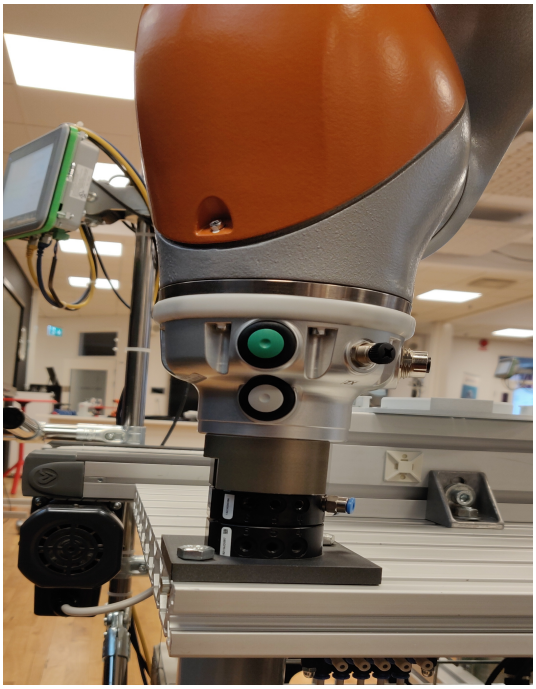
### 3.5.2 Robot programming

The KUKA iiwa robot used in the project was programmed using offline programming method as explained in chapter 2, section 2.4.5. KUKA sunrise workbench was the external software application that was used to write the robot program in java coding. The tool coordinate points for robot movement were retrieved from simulation platforms and added in application data tab of Sunrise workbench. The coordinate points for the robot movements are provided in Appendix D, table D.1. These points were obtained after matching the base and tool calibration values in simulation platforms to increase accuracy of the assembly process. The process followed is provided below.

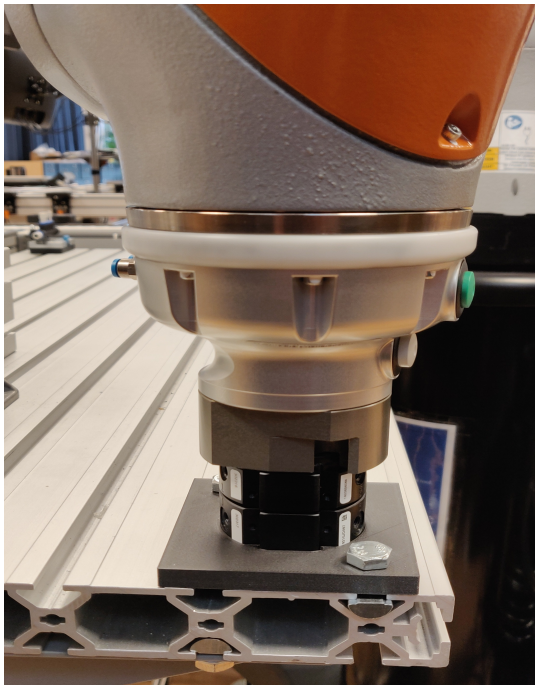
- The base calibration values were obtained from the KUKA smartPAD post calibration procedure.



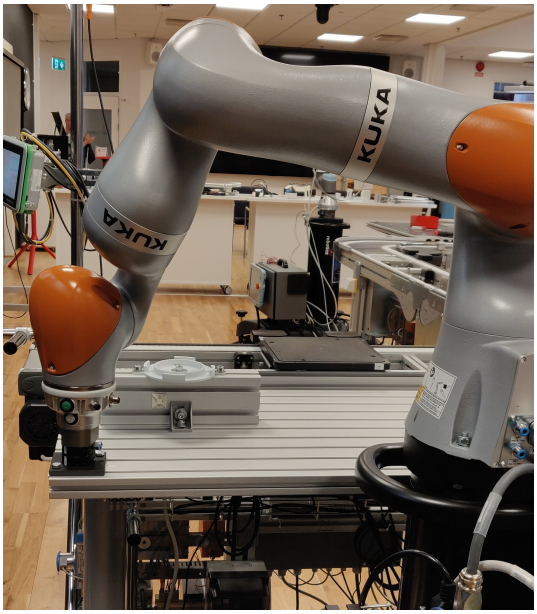

(a) Hand-guiding movement of KUKA robot



(b) Placement on tool adaptor



(c) Check for orientation




(d) Record cartesian coordinates

**Figure 3.6:** Hand-guiding method

- Since the left bottom corner of the table was used as reference in 3-point base calibration procedure, the working frame on the simulation platform was also set as the same.
- Once the working frame is set on the simulation platform, yet another frame was created.
- The relative position of the new frame with respect to working frame was set with the base calibration values adhering to the direction and orientation.
- The robot with gripper and base were moved by choosing the robot baseframe and selecting the relocate option.
- The selection was moved to the new frame created.
- The distance between the working frame and new position of the robot baseframe was measured.
- This measurement was equal to the base calibration values.
- Thus, the calibration values were matched in the simulation platform.
- The coordinate points were updated in the simulation values with respect to the working frame.
- The coordinate values in physical setup are considered with respect to the base coordinates.

The media flange used in this KUKA iiwa robot is a touch pneumatic media flange. Touch pneumatic media flange is a universal interface to enable users to connect pneumatic and electrical components to robot flange [34]. The media flange signals were connected to Schunk gripper and the connection diagrams are shown in figure 3.7. The media flange signals were used for gripper actions like open/release, close/grasp, setting LED lights and switching voltage. Finally, the robot program was synchronized with robot controller and executed on the KUKA smartPAD.

### 3.5.3 Force torque analysis

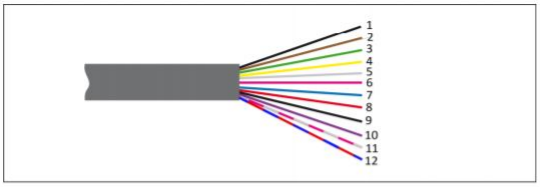
The KUKA iiwa robot is equipped with position and joint torque sensors and was earlier mentioned in section 2.4.7. The sensors were used to record force and torques generated at the Tool centre point (TCP) of schunk gripper along with joint external torques at various instances. The sensors were triggered at certain coordinate points during assembly of motor with drone frame or during picking of materials from pallets. The values recorded at the events were used in analyzing the variation of forces or torques acting on the robot during the following instances provided below.

- Assembly of motors with *correct/incorrect* orientation.
- *Availability/unavailability* of drone components on the pallet during pickup.

Changes in the robot algorithm were made to implement alternate operations in the case of discrepancies.

#### Sensitivity analysis

An additional experiment was conducted on the KUKA robot to test the accuracy and sensitivity of the sensors. The schunk gripper was dismantled and known

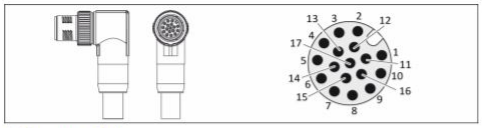



*Cable with 12 wire strands*

*Connection assignment*

| No. | Wire strand | Signal |
|---|---|---|
| 1 | White | Digital input DI 2 |
| 2 | Brown | Digital input DI 1 |
| 3 | Green | Sensor 2 |
| 4 | Yellow | Sensor 1 |
| 5 | Grey | Sensor 4 (optional, on request) |
| 6 | Pink | Sensor 3 (optional, on request) |
| 7 | Blue | Closes gripper |
| 8 | Red | Opens gripper |
| 9 | Black | n.c. |
| 10 | Violet | n.c. |
| 11 | Grey/Pink | GND |
| 12 | Red/Blue | +24 VDC |

**(a)** MFIO wire info




*M12 connector, 17-pin*

*Connection assignment*

| Pin | Wire strand | Signal | KUKA Connector X3 |
|---|---|---|---|
| 1 | Brown | Closes gripper | MF_Output_0 |
| 2 | Blue | Digital input DI 2 (LED light band) | MF_Output_3 |
| 3 | White | Sensor 4 (optional) | MF_Input_3 |
| 4 | Green | n.c. | MF_Input_4 |
| 5 | Pink | GND | GND_24V |
| 6 | Yellow | n.c. | GND_24V |
| 7 | Black | +24 VDC | Output_VCC |
| 8 | Grey | n.c. | Output_VCC |
| 9 | Red | n.c. | Output_VCC |
| 10 | Violet | Sensor 1 | MF_Input_0 |
| 11 | Grey/Pink | Opens gripper | MF_Output_1 |
| 12 | Red/Blue | Digital input DI 1 (LED light band) | MF_Output_2 |
| 13 | White / Green | Sensor 3 (optional) | MF_Input_2 |
| 14 | Brown / Green | n.c. | GND_24V |
| 15 | White / Yellow | n.c. | Output_VCC |
| 16 | Yellow / Brown | Sensor 2 | MF_Input_3 |
| 17 | White / Grey | n.c. | GND_24V |

**(b)** MFIO pin info

**Figure 3.7:** Media flange connection information [34]

weights were attached to the robot flange. The robot was moved to predefined coordinate points. The force/torque acting at the flange were recorded at a rate 100 ms for comparison. The cartesian forces and torques were also recorded and displayed on the smartPAD when the robot reached the coordinate points. The orientation of the flange coordinate axis was also considered during experimentation. Several trials were conducted with various weights at the robot flange.

Results from the experiment were tabulated and plotted. The results and observations will be presented in the subsequent chapter.

## 3.6 Logical control programming

This stage involved designing of logical device behaviours and system control models. Logical device modeling included the simulation application interfaces that were linked to the control logic models. The control logic models involved programming of PLCs. The methodology followed is explained in this section. This stage could also be performed in parallel with the simulation stage to improve the process timeline.

### 3.6.1 PLC programming

PLC programming was performed on Siemens TIA portal V15.1 application for this thesis project. The application was launched with a new project filename `IMSX30_VirtualCommissioning.ap15_1`. The choice of controller was CPU 1211C AC/DC/Rly configuration from SIMATIC S7-1200 family. The PLC program was

created in ladder logic within Main [OB1] at the program block tab using different operators such as bit logic and timer operators.

PLC tags tab was opened to provide data types, address with names that were used on the ladder logic. The logic consisted of a normally closed contact that activates the output initially. The normally open contact connected in parallel when true, deactivates the output. During this period, the KUKA iiwa robot performs the assembly operation and sends a signal once the process is complete. This ladder logic was compiled to check for errors and prepared for Software in loop (SiL) testing along with simulation applications by loading the programmable logic on to the virtual controller. The process for the same is mentioned in the section below.

## 3.7 Virtual commissioning with SiL method

After all stages were successfully completed, Virtual Commissioning process was performed. Software in loop (SiL) method was followed in this thesis project. Chapter 2, section 2.6.2 provides a brief explanation of the process. Digital representation of the real station was prepared on the simulation software and programmable logic was tested and mapped to the simulation environment. The procedure followed using Visual Components software is explained in the below section.

### 3.7.1 Visual components

Virtual Commissioning with SiL method was conducted using Visual Components application by connecting to a virtual PLC controller. This was performed by connecting TIA portal V15.1 with S7-PLCSIM V15.1 application. Since PLC SIM can not be directly connected to Visual Components, NettoPLCsim application was used to connect Visual Components with S7-PLCSIM V15.1. The complete procedure for the same is provided in Appendix H, section H.1.1 and the same was obtained from Visual Components academy [35].

# 4
# Results

The results achieved during the project work are provided in this chapter. The results are presented in sections, similar to the process followed in chapter 3.

## 4.1 Data collection

The results from this stage are presented in the section. The process involved is explained chapter 3, section 3.1.

### 4.1.1 Literature research

The application of Virtual Commissioning in a manufacturing system with mass customization and oscillating demands is clearly presented by Mortensen et al. [29]. The paper motivates the use of novel technologies like Virtual Commissioning, that are being tested at learning factories. SII-Lab is also one such setup. The paper also discusses the design procedure for Virtual Commissioning. A concurrent design methodology is proposed by Ko et al., in their article for Virtual Commissioning projects [31]. The article discusses the shortcomings faced in the conventional design procedure of a production system with DES software like AutoMod® or ARENA®, where the control logic cannot be tested. Thus, motivating the importance of testing virtual models with PLC, a concurrent methodology is proposed. Dumitrascu's article also discusses the importance of Virtual Commissioning and presents various configurations for implementation like Software in loop (SiL) and Hardware in loop (HiL) [32]. The article further discusses the prerequisites and sheds light on Tecnomatix Process Simulate as a probable software application. The benefit in workflow with Virtual Commissioning is presented in an article on the Visual Components website [30]. A similar motivation in Lechler's article is presented that exhibits the benefit of Virtual Commissioning in an engineering process [36]. The economical justifications provided by Shahim in their article proves the value of Virtual Commissioning in the industrial setup [33]. Master thesis projects [37, 38, 39] presented at the university were also investigated. Thus, the methodology for Virtual Commissioning project is drafted as presented in figure 2.11.

The KUKA sunrise manual provides us with details regarding the KUKA LBR iiwa robot and advantages of 7-axis joints that improves the reachability and maneuverability of the robot. The presence of sensors at the joints motivated us to conduct the force/torque analysis.

### 4.1.2 Technical data

The drone factory layout information was obtained from SII-Lab and is provided in the below figure 4.1. The CAD parts of the drone components, fixture, robot and pedestal etc obtained are provided in Appendix E, section E.1.1. The collected data aided in developing the layout on simulation platforms post software selection stage, presented in the below section. This stage is of extreme importance, as latest revisions of the CAD models need to be obtained. Several improvements are conducted throughout the processes. Thus, updating CAD data and documenting the revision numbers becomes an important requirement.

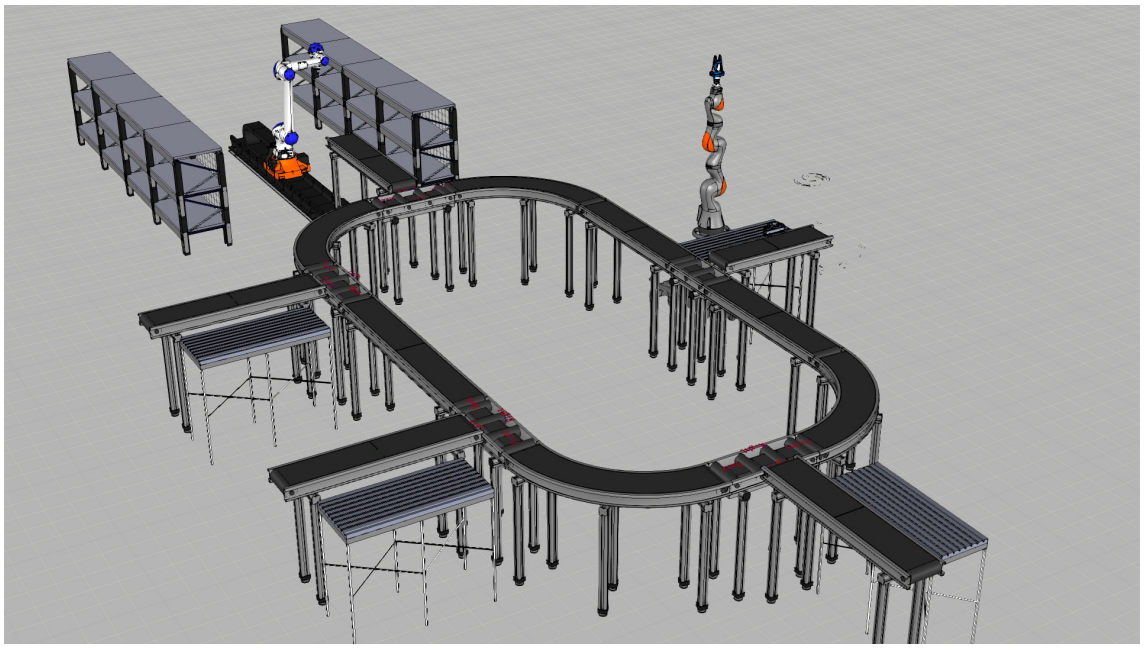


**Figure 4.1:** Layout arrangement at SII-Lab

### 4.1.3 Software selection

With due consideration to all the features offered, the following simulation software applications were shortlisted.

1. Visual Components premium 4.2 by Visual Components®
2. Tecnomatix Process Simulate by Siemens®

Considering the features offered by both Visual Components and Siemens, simulation of drone assembly process is decided to be carried out on both software applications. Utilization of both the software applications in the project assisted in gauging the functionality and robustness across platforms and in-turn aid in answering the research question of best software platform for Virtual Commissioning.

The modeling software chosen to access the CAD files and perform assemblies is Catia V5 2019.

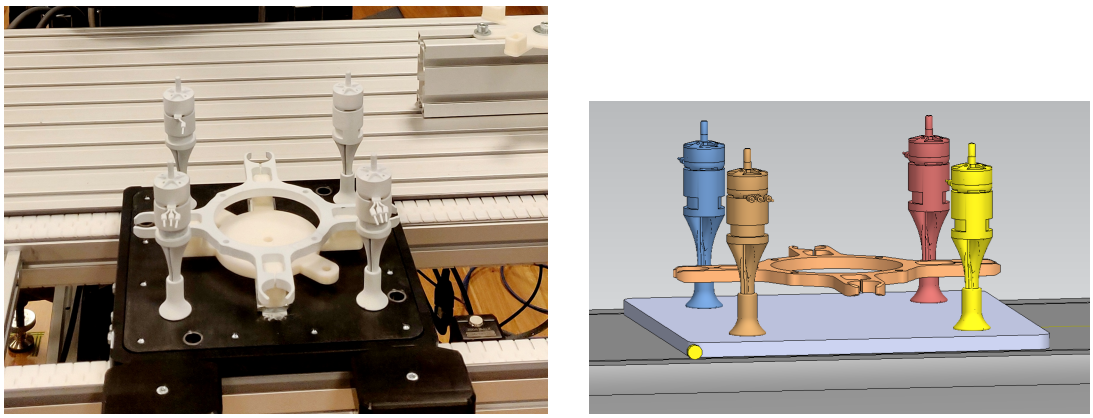
The modelling software is used to assemble parts of the gripper and export to the simulation software. Since Tecnomatix Process Simulate was not linked to online libraries, required models were exported from Visual Components to the required format before importing on Tecnomatix Process Simulate.

## 4.2 Process planning

The steps followed in this stage are explained in chapter 3, section 3.2. Process planning in robotic based assembly systems, presented by Browne is discussed previously. The procedure is effective and proved to be ideal in our situation. However, the economic feasibility factor is not considered. The methodology adopted clearly exhibits the iterative process among the stages and improvements in every iteration to achieve an ideal condition. The results from the stage are presented below.

### 4.2.1 Component placement on pallet

The drone components placed on the pallet is shown in figure 4.2. The drone frame is placed at a height of approximately 25mm from the pallet to avoid collision of gripper with pallet. This is managed by placing the drone frame on holding fixture as displayed in figure 4.2a. The same is managed on the simulation platform through frame management by increasing coordinate value in the z direction during simulation process.



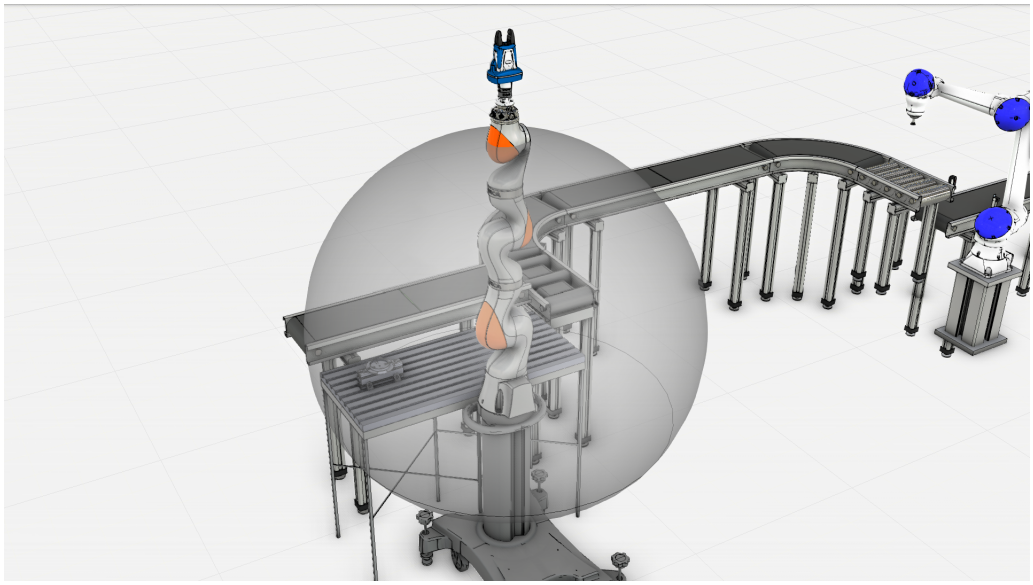
**(a)** Physical station **(b)** Simulation platform

**Figure 4.2:** Component placement

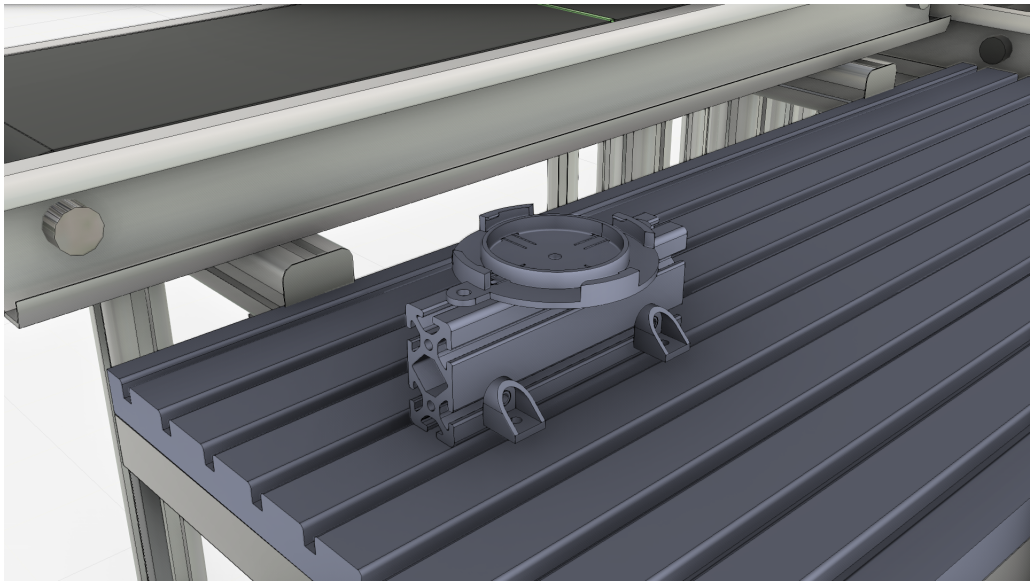
### 4.2.2 Robot and fixture placement

Robot is decided to be placed in front of the station as per the factory layout available at SII-Lab. The reachability of the robot is briefly tested by conducting a reach test on Tecnomatix Process Simulate. Similarly, on Visual Components, it is performed by selecting 3D envelope option as shown in figure 4.3a. However, the

layout is prepared accurately in the layout preparation stage as discussed in section 4.4. Additionally, the KUKA iiwa robot has a maximum reach of 820mm as provided in table 2.3a. Several positions for placement of fixture were tested and checked for reachability and prevention of redundancy of the KUKA robot on the simulation platforms. The fixture is finally decided to be placed on the table horizontally at the top left corner as displayed in figure 4.3b. Further, the exact location is calibrated during the simulation process.



**(a)** Robot 3D workspace



**(b)** Fixture placement

**Figure 4.3:** Robot and fixture placement

### 4.2.3 Assembly operation sequence

The assembly operation sequence for the drone is presented in figure 4.4. The fixture is designed to hold the drone frame. Hence, the first operation in the assembly process after arrival of components, is the frame assembly on to the fixture. Further, the motors are assembled. The order of motor assembly is designed in the simulation stage considering the robot reach and joint angles at the fixing positions.

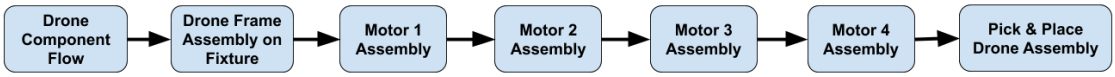


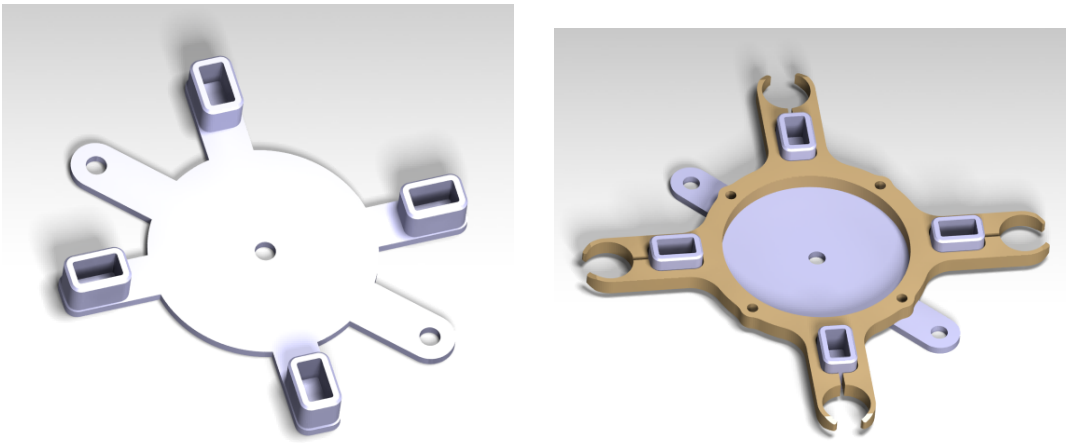

**Figure 4.4:** Assembly operation sequence

## 4.3 3D Modelling

The results obtained from the stage are provided in this section. This stage involved modeling of components for defining kinematics during simulation of the drone assembly process. The methodology followed is explained in section 3.3.

The CAD model part files were launched on CATIA V5 and assembled with constraints on the assembly workbench. Schunk CAD parts and resulting CAD product is provided in Appendix E, section E.2.1. The process also aided in designing new fixture for drone assembly and testing gripper finger modifications. The parts were 3D printed and assembled on the physical gripper for testing the solutions. Emphasis is to given to the design improvements of the components in this stage. Documenting the revision numbers are deemed of utmost importance and several trials are conducted in tandem with process planning and simulation stage. The components designed on Catia V5 should be incorporated with a structured specification list and geometrical sets for each part. Additionally, The CAD models are to be designed in due consideration of the coordinate axis. These features aid in the simulation stage while defining the kinematics of the components.

Figure 4.5 depicts the design improvement on the drone fixture. This process aided in improving the cycle time of the assembly process through reduction in robot coordinate points. The resuts from the simulation stage are presented in the below section.



**(a)** New fixture design **(b)** Fixture with drone frame

**Figure 4.5:** Design improvements

## 4.4 Simulation

The results obtained are explained in the section. The results are categorized as individual sections for each software application used in the process i.e., Visual Components (VC) and Tecnomatix Process Simulate (TPS). The process followed in this stage is provided in chapter 3, section 3.4.

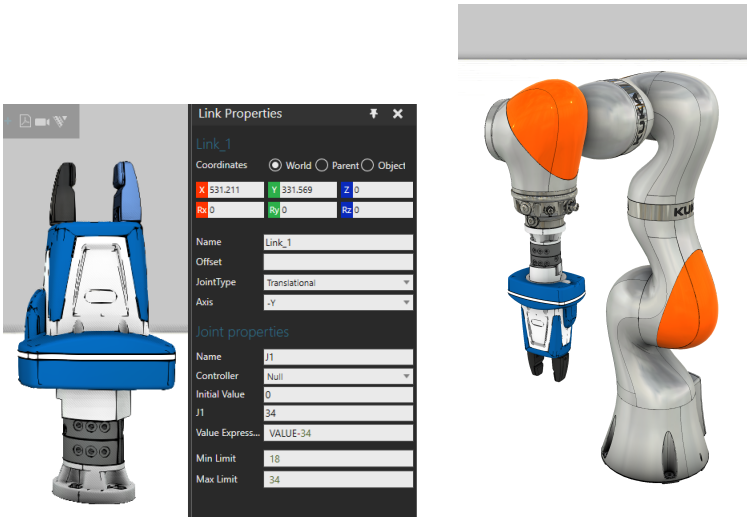
### 4.4.1 Visual components

#### Model a component

The process was a requirement as the gripper used for the project was not available in Visual Components library. The method adopted for modeling the gripper is explained in the previous chapter. Figure 4.6, shows the results of the process. Signals are set for grasp and release actions of the gripper in this stage. The modeling process aids in defining the kinematics of the components and further, activates the PnP option. The Schunk gripper is assembled on the robot as shown in figure 4.6b. The KUKA iiwa robot is further assembled on the robot positioner, that is modeled similarly and prepared for layout setup.

#### Layout preparation

The layout as available at SII-Lab, is prepared with operative modifications. The availability of parameterization aids in altering the dimensions of the components according to the requirements. Drag and drop along with PnP option shortens the layout preparation time. Additional options like align and snap could also be used. Thus, improvements in the layout can be performed easily with the aforementioned features. The figure 4.7 below, depicts the layout created on Visual Components. The next step involved simulating the assembly process whose results are provided below.




**(a)** Gripper model

**(b)** Robot with gripper

**Figure 4.6:** Schunk gripper modeling on Visual components

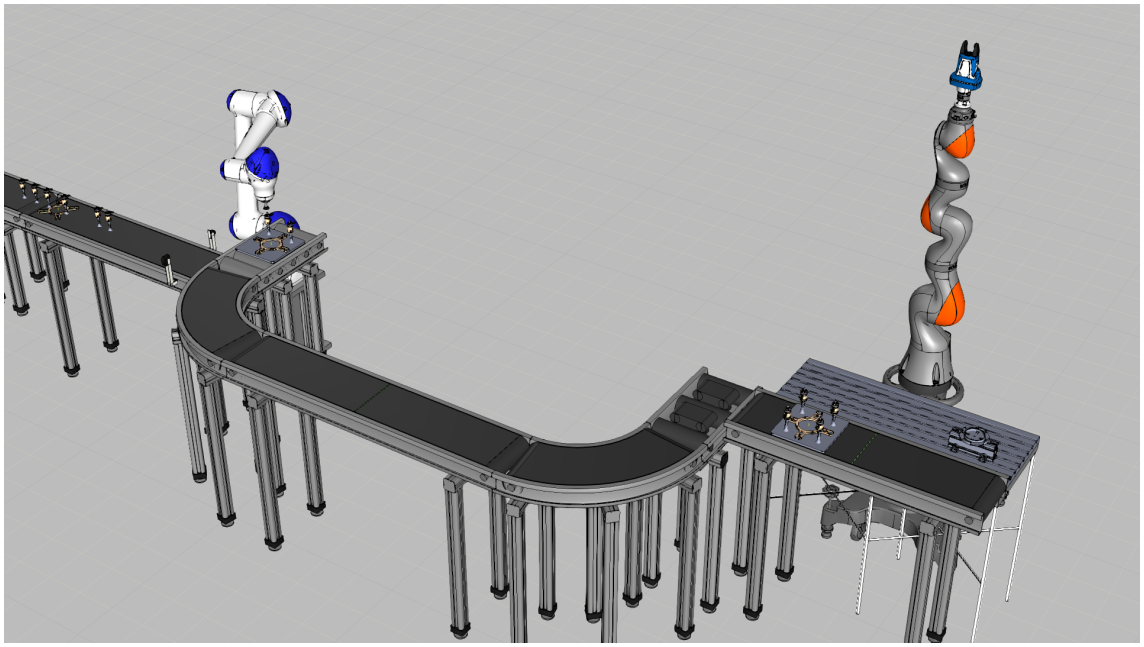


**Figure 4.7:** Layout model on Visual Components

#### Assembly process simulation

The robots are programmed to conduct the assembly of drone frame with motors. Motoman robot is used to pick and place drone components onto pallets and further, KUKA iiwa robot is used in the drone assembly process. Coordinate points are defined for the robot using Linear and Point to Point motions. The coordinate points with respect to world or parent object can be found on the properties window. The robot joint angles are also displayed during the process that aids in determining the range and turn. Figure 4.8 below is a snapshot of the assembly process simulated on Visual Components. The simulation process aided in defining the robot motions by avoiding collisions and determining the cycle time of the process. Iterative improvements are made to reduce the cycle time and perceive the safety requirements of the cell ahead of physical commissioning.

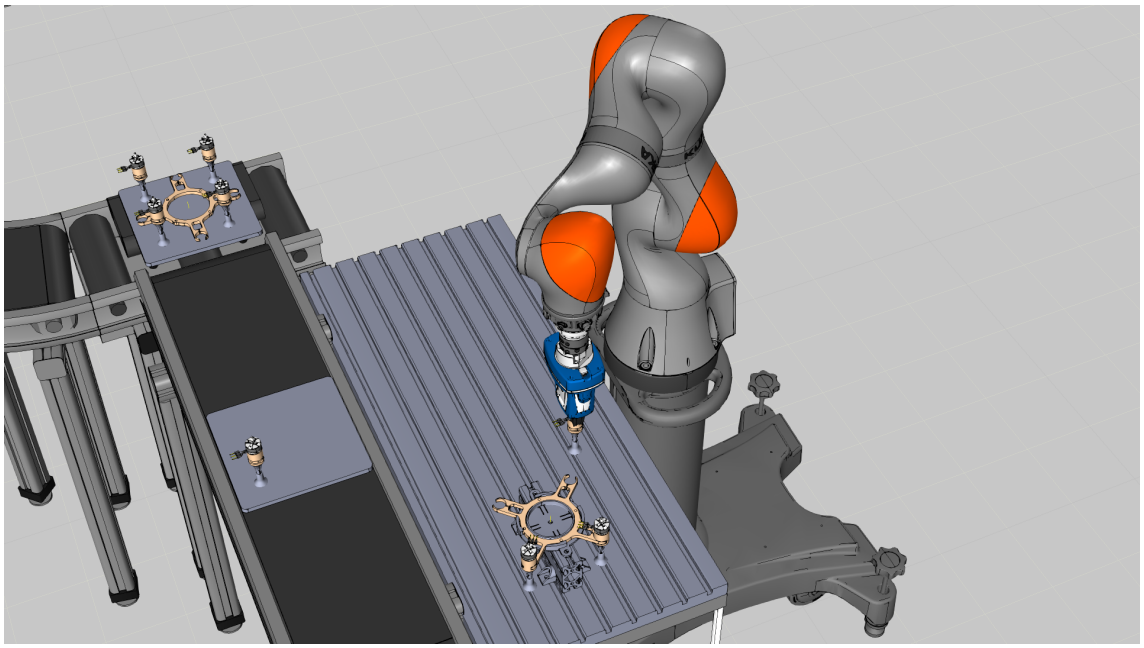
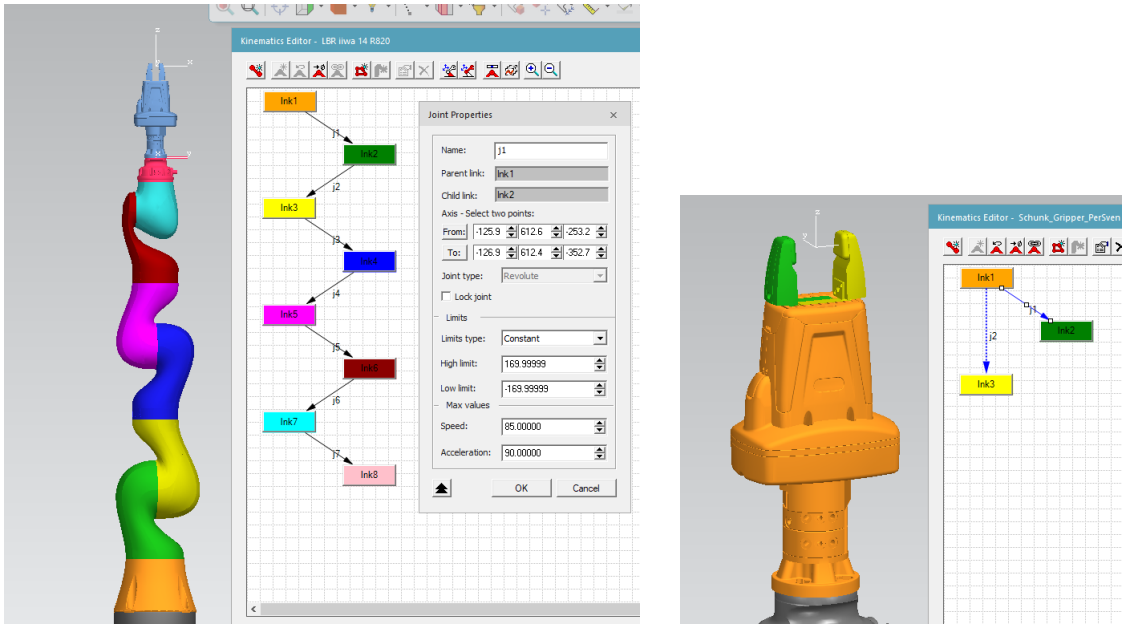


**Figure 4.8:** Assembly simulation on Visual Components

### 4.4.2 Tecnomatix process simulate

#### Setting kinematics

Kinematics for components like robot and gripper are established and the process is explained in the previous chapter. Figure 4.9 shows kinematics editor results of KUKA iiwa robot and Schunk gripper. After setting the base coordinates and tool coordinates for both components, the gripper is mounted on the robot by choosing Mount Tool option as shown in figure 4.10. The next step is setting the layout and the same is discussed below.




(a) KUKA kinematics (b) Schunk gripper kinematics

**Figure 4.9:** Kinematics results

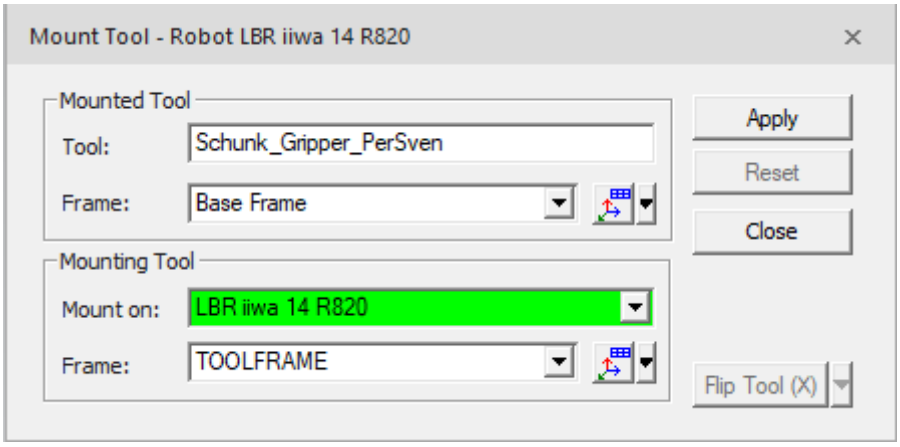



**Figure 4.10:** Mount tool option

### Layout preparation

Layout preparation procedure is briefly explained in chapter 3 and the results of the same are presented here. The components imported as parts/resources are arranged according to the layout at SII-Lab adhering to calibration values obtained during the physical setup stage. A major disadvantage with the unavailability of online library is parameterization of the components. Dimensional changes to the components are performed on Visual Components, and imported. Setting of kinematics can be complicated as individual parts of the imported models cannot be separated. Any improvements to CAD models were to be converted on Visual Components and imported as well. This made the process time consuming and complicated. Figure 4.11 shows the layout arrangement created in Tecnomatix Process Simulate.

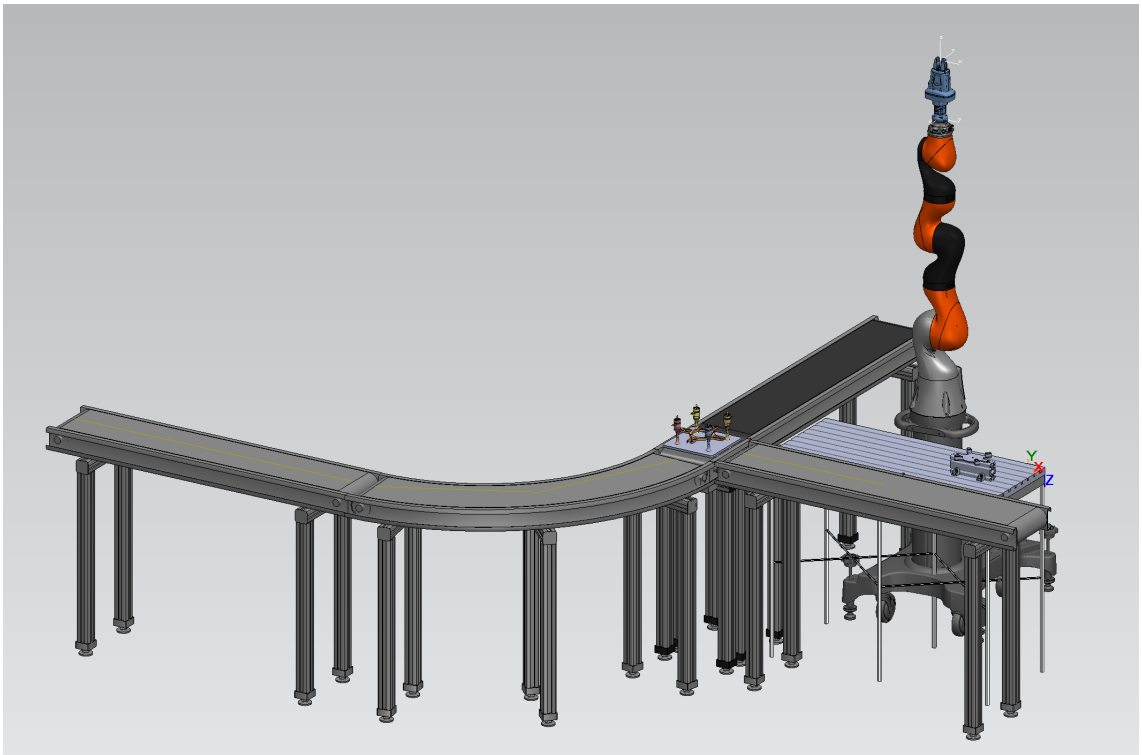


**Figure 4.11:** Layout model on Tecnomatix Process Simulate

### Assembly process simulation

Simulation of drone assembly is performed as per the procedure explained in the previous chapter. The process aided in ideal placement of the robot and test its reach as well. Robot movements are planned to reduce cycle time without collisions. The inputs from these tests are used in process planning and modeling stages, thus demonstrating the iterative and continual information sharing within the processes. The frame management feature aids in calibrating the base and tool coordinates as per physical setup, thus improving the accuracy and resemblance in the simulated environment. The path editor panel in the main window furnishes the coordinate points. These coordinate values from the simulation process are used in

robot programming. Figure 4.12, provides a clipping from the simulated assembly process on Tecnomatix Process Simulate.

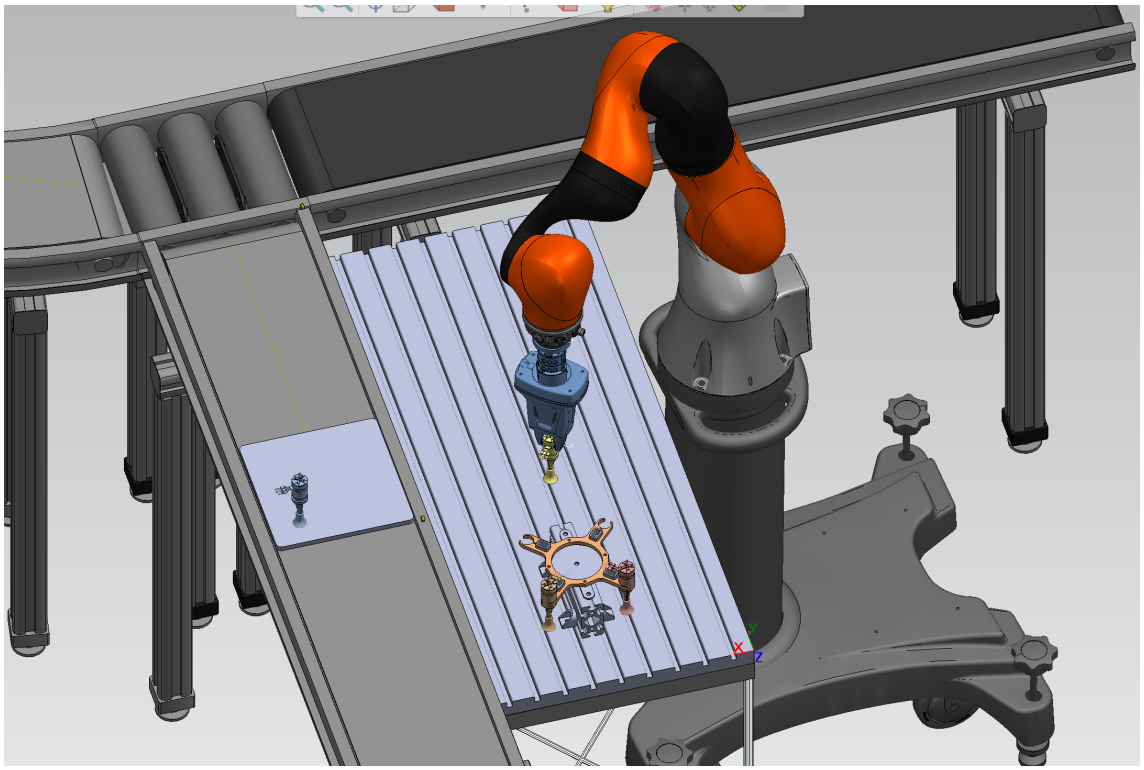


**Figure 4.12:** Assembly simulation on Tecnomatix Process Simulate

## 4.5 Physical implementation and verification

This section presents the results obtained from physical verification and implementation stage. The methodology followed in this stage is explained in the previous chapter.

### 4.5.1 Robot calibration

The results obtained from calibration procedures performed on the KUKA iiwa robot are discussed in this section. The methods followed in calibrating the robot base and tool are explained in section 3.5.1. The calibration values were synced from the robot smartPAD to KUKA sunrise workbench after completion.

#### Tool calibration

Tool calibration results are presented in this section. The methods followed to define the origin and orientation of tool coordinate system are explained in chapter 3. The procedure was carried on both Schunk gripper and pointer tool. The pointer tool was used in the base calibration process. The results from the procedure are provided in table 4.1.

**Table 4.1:** Tool calibration results

**(a)** Schunk gripper

| **Origin values (in mm)** | |
|---|---|
| X | -1.41 |
| Y | -1.13 |
| Z | 282.08 |
| **Orientation (in degrees)** | |
| A | 89.08 ° |
| B | 1.84 ° |
| C | 1.27 ° |
| **Calculation error (mm)** | |
| 0.51 | |

**(b)** Pointer tool

| **Origin values (in mm)** | |
|---|---|
| X | -0.32 |
| Y | -1.27 |
| Z | 165.19 |
| **Orientation (in degrees)** | |
| A | 178.08 ° |
| B | 0.11 ° |
| C | -2.32 ° |
| **Calculation error (mm)** | |
| 0.37 | |

### Base calibration

The results from base calibration procedure of KUKA iiwa robot are presented in this section. The procedure followed was explained in section 3.5.1. The robot is mounted with a pointer tool during the base calibration method for increased accuracy of the results. The calibration values are then conformed on the simulation applications to improve coordinate point accuracy during robot programming. The procedure for the same is provided in section 3.5.2. The base calibration values obtained, are synced to KUKA sunrise workbench from the KUKA smartPAD after the procedure. The results from the 3-point base calibration method are presented in table 4.2.

**Table 4.2:** Base calibration results

| **Origin values (in mm)** | |
|---|---|
| X | 124.1 |
| Y | 612.8 |
| Z | -107.7 |
| **Orientation (in degrees)** | |
| A | -0.5 ° |
| B | 0.6 ° |
| C | 179.9 ° |

### Hand-guiding method

The results obtained from handguiding method are presented in this section. The procedure and aim of this method are explained in the previous chapter. The experiment proved that hand-guiding method, if performed without base calibration would provide the cartesian coordinates with respect to the world coordinate system and can be directly added as base coordinates in the application data of Sunrise

workbench. Table 4.3, provides the flange position from base origin obtained during the test. The base coordinates are obtained previously using the 3-point method. The deviation obtained is matched in the simulation platform to further increase the accuracy of the robot coordinates. Thus, the hand-guiding feature on the KUKA iiwa robot is effective and can be used in locating the accurate position of the base origin.

**Table 4.3:** Hand-guiding experiment results

| **Flange position from base origin (mm)** | |
|---|---|
| X | 0.62 |
| Y | 2.13 |
| Z | 2.08 |

### Variation analysis

The results from the variation analysis experiment is presented in this section.

Base calibration of the KUKA iiwa robot is conducted using the 3-point method initially and the results are tabulated. Additionally, the flange coordinates with respect to the world coordinate system is determined using the hand-guiding method. The values of the same are tabulated as hand-guiding coordinates. Further, the position of the robot is altered and the calibration procedures are repeated. Table C.1 in Appendix C, presents the tabulated base calibration results from 13 different positions. Figure 4.13, represents a scatter plot of the base coordinates obtained at 13 positions. Table 4.4, provides the standard deviation of the absolute difference of base coordinate values, obtained from the variation analysis experiment.

**Table 4.4:** Standard deviation of absolute difference

| **Standard deviation (s)** | |
|---|---|
| X | 0.829 |
| Y | 0.889 |
| Z | 0.561 |

The results of this experiment can further be analyzed by the following method. A physical point in the station is chosen, eg. centre of the drone fixture. The robot can be moved to the chosen point after 3-point calibration procedure and the pointer tool TCP coordinates with respect to the base coordinate system can be recorded. Further, the above mentioned process is repeated after the hand-guiding calibration method. The variation in the coordinate values after both the methods can be tabulated to evaluate the standard deviation, thus, determining the robustness of the KUKA iiwa robot.

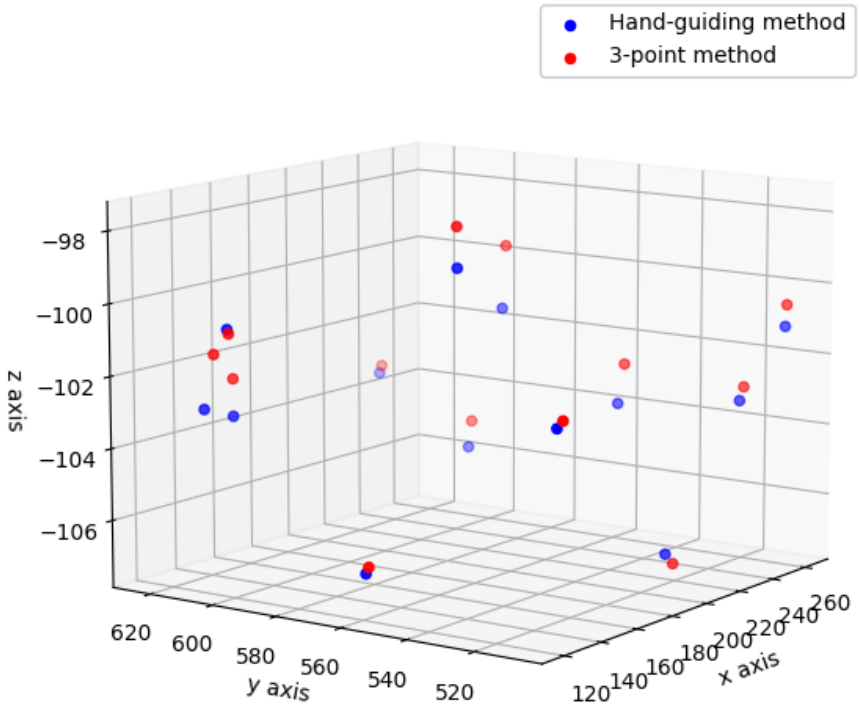



**Figure 4.13:** Scatter plot of Base coordinates

### 4.5.2 Robot programming

The coordinates for robot programming are obtained from simulation applications. The values obtained from the simulation models are entered manually into KUKA sunrise workbench in application data. Media flange signals to the Schunk gripper were injected into the robot program for grasp and release operations. The base and tool calibration values obtained from KUKA smartPAD were matched in simulation applications to increase the accuracy of the assembly process. Figure 4.14, provides the result of the same in the simulation platform. A statement from the source code is provided below that executes the robot movement towards position via3 with respect to the defined base.

```
Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via3")));
```

Finally, the robot program was synchronized with the robot controller and executed on KUKA smartPAD. The complete source code for the robot assembly process and media flange is provided in F, section F.1. Additionally, the pictures of the assembly process executed at SII-Lab, is provided in section F.2.

### 4.5.3 Force torque analysis

Real-time force and torque values for the KUKA iiwa robot can be obtained from the smartPAD during robot motion. These values can be used as conditions while writing the robot program to trigger certain actions or to identify any variations in the process. For this drone assembly process, force and torque values are measured for 2 instances as explained in section 3.5.3. External cartesian forces at TCP and

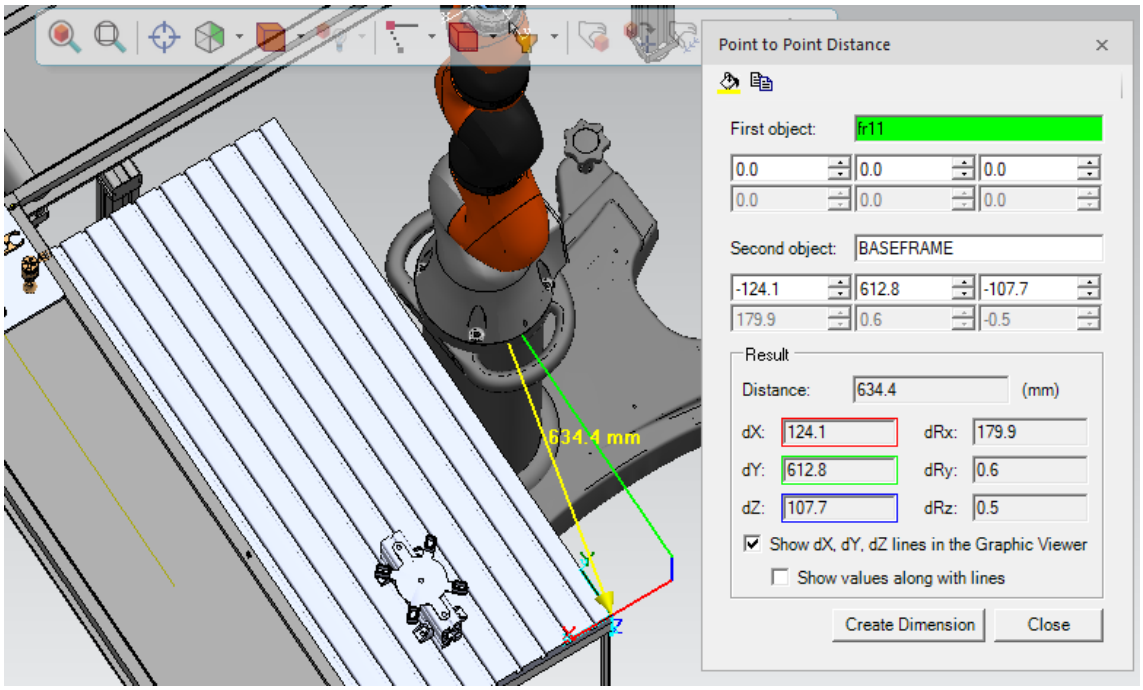



**Figure 4.14:** Matching calibration in Simulation

external joint torques at each joint are measured during picking of drone frame to check the availability and during assembly of motor on to the frame to check the orientation.

The data recording option is initialized and activated to record the external joint torques and cartesian forces acting on the gripper TCP for a duration of 150 seconds with a recording rate of 50 ms [14].

```
DataRecorder rec1 = new DataRecorder("Recording_16June.log", 150, TimeUnit.SECONDS, 50);
rec1.addExternalJointTorque(lBR_iiwa_14_R820_1);
rec1.addCartesianForce(Gripper.getFrame("/GripperTCP"), null);
rec1.enable();
```

The data displayed on smartPAD using 'logger.info' option as shown is used to establish the conditions and set of operations.

```
//Force and Torque data is displayed
ForceSensorData data = lBR_iiwa_14_R820_1.getExternalForceTorque(Gripper.getFrame
("/GripperTCP"));
TorqueSensorData measuredata = lBR_iiwa_14_R820_1.getMeasuredTorque();
logger.info("Force & Torque :" +data);
logger.info("Measured Torques :" +measuredata);
```

The force and torque component conditions were initialized and inverted to set the range. Motion command is temporarily stored using the `IMotionContainer` command. Motion termination is performed using a `breakWhen()` command when the conditions initialized are met. Information of terminated motions are stored in `IFiredCondtionInfo`. The condition which caused the termination of a motion can be requested via the method `getFiredCondition()`. If the requested information is not equal to null, the motion will be terminated. The system only requests the

triggered break condition in our case. Specific operations are provided in the source code for the robot to perform. The orientation of the gripper TCP, while the sensors are trigger is given in figure 4.15, where '↓' is z axis, 'I' is y axis and ' • ' is x axis. The source code with conditions and triggers for different scenarios is provided in Appendix G, section G.2. A graphical comparison of the measured values for tried instances is shown in Appendix G.

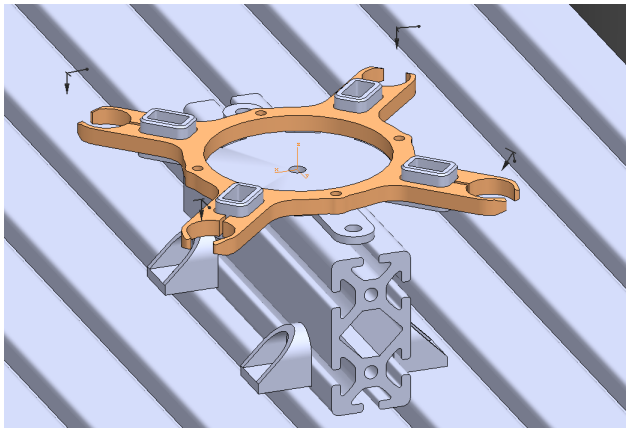


**Figure 4.15:** Gripper TCP orientation

The KUKA iiwa robot is capable of measuring small variations in force and torque values, the measured force and torque values obtained during the aforementioned test scenarios, shows that the values do not vary significantly to observe any change. Also, the measured values have significant overlap with every trial, thus resulting in difficulty to differentiate between correct and wrong processes. The probable reasons for these variations are explained in section 2.4.7. Since the values were inconsistent, the sensitivity analysis of the robot sensors are performed. The results are presented below.

### Sensitivity analysis

The external joint torque and cartesian forces acting on the flange TCP are recorded at a rate of 100ms and also at coordinate points. A sample source code is provided in the previous section. The recorded values are tabulated and shown in Appendix G.3, table G.1. The presented values also include the weight of the gripper and pointer tool. The values are also plotted to observe the variation. Appendix G.3, provides the results of the plots. The coordinate points for the robot movement are programmed in the following conditions.

1. X axis of the flange pointing down (Coordinate point 1)
2. Y axis of the flange pointing down (Coordinate point 2)
3. Z axis of the flange pointing down (Coordinate point 3)

From table G.1, the average value of $F_x$ with no weight (0 grams), excluding the gripper and pointer tool weights recorded is $F_x$ = 4.64 N. Similarly, the average

with 1350 grams weight recorded, is $F_x$ = 17.09 N. To verify the value, following calculations are performed.

$$F_{x\ (1760)} = F_{x\ (0)} + (g \times m)$$

(g x m as force is acting in downward direction)

$$F_{x\ (1760)} = 4.64 + (9.81 \times 1.350)$$

$$F_{x\ (1760)} = 17.8N$$

Similar calculations can be performed to test other values of cartesian force and torques. The weight of the adaptor hubs are approximately 480 grams and they contribute to the force/torque acting on the flange. Hence, $F_{x\ (0g)}$ = 4.64 N is obtained. The values tend to change upon addition of weights as shown above. Thus, it can be proved that KUKA iiwa robot is capable to detecting even small variations of forces and torques acting externally. The graphs plotted through continuous recording also prove the same. Further results can be obtained by continuously recording cartesian torques acting at the TCP using `rec.addCartesianTorque();` feature.

## 4.6 Logical control programming

This stage involves the design of programmable logic. PLC programming was performed on TIA portal V15 by Siemens. The programming logic is loaded on to a virtual control S7PLCSIM application. This setup is then verified by Virtual Commissioning (VCom) method as explained in section 3.7. The results from the electrical stage is provided in the sections below.

### 4.6.1 PLC programming

The logic on TIA portal is prepared in a way that conveyor motor is active until pallet with drone components activates the conveyor sensor, in-turn stopping the conveyor motor. The KUKA iiwa robot now performs the drone assembly operation. A pneumatic compressor is activated when the gripper grasps the drone components and is deactivated otherwise. After the assembly operation is complete, the robot sends a signal and the conveyor motor is reactivated. The CPU controller information is provided in section 3.6.1. PLC ladder logic along with tags allocated on TIA portal are provided in figure 4.16 and table 4.5.

**Table 4.5:** PLC Tags

| Name | Path | Data Type | Logical Address | Hmi Visible | Hmi Accessible | Hmi Writeable |
|---|---|---|---|---|---|---|
| Conveyor_Sensor | Default tag table | Bool | %I0.0 | True | True | True |
| Conveyor_Motor | Default tag table | Bool | %Q0.1 | True | True | True |
| Gripper_OPEN | Default tag table | Bool | %I0.2 | True | True | True |
| Pneumatic_System | Default tag table | Bool | %Q0.4 | True | True | True |
| Gripper_CLOSE | Default tag table | Bool | %I0.3 | True | True | True |
| Robot_Input | Default tag table | Bool | %I0.6 | True | True | True |

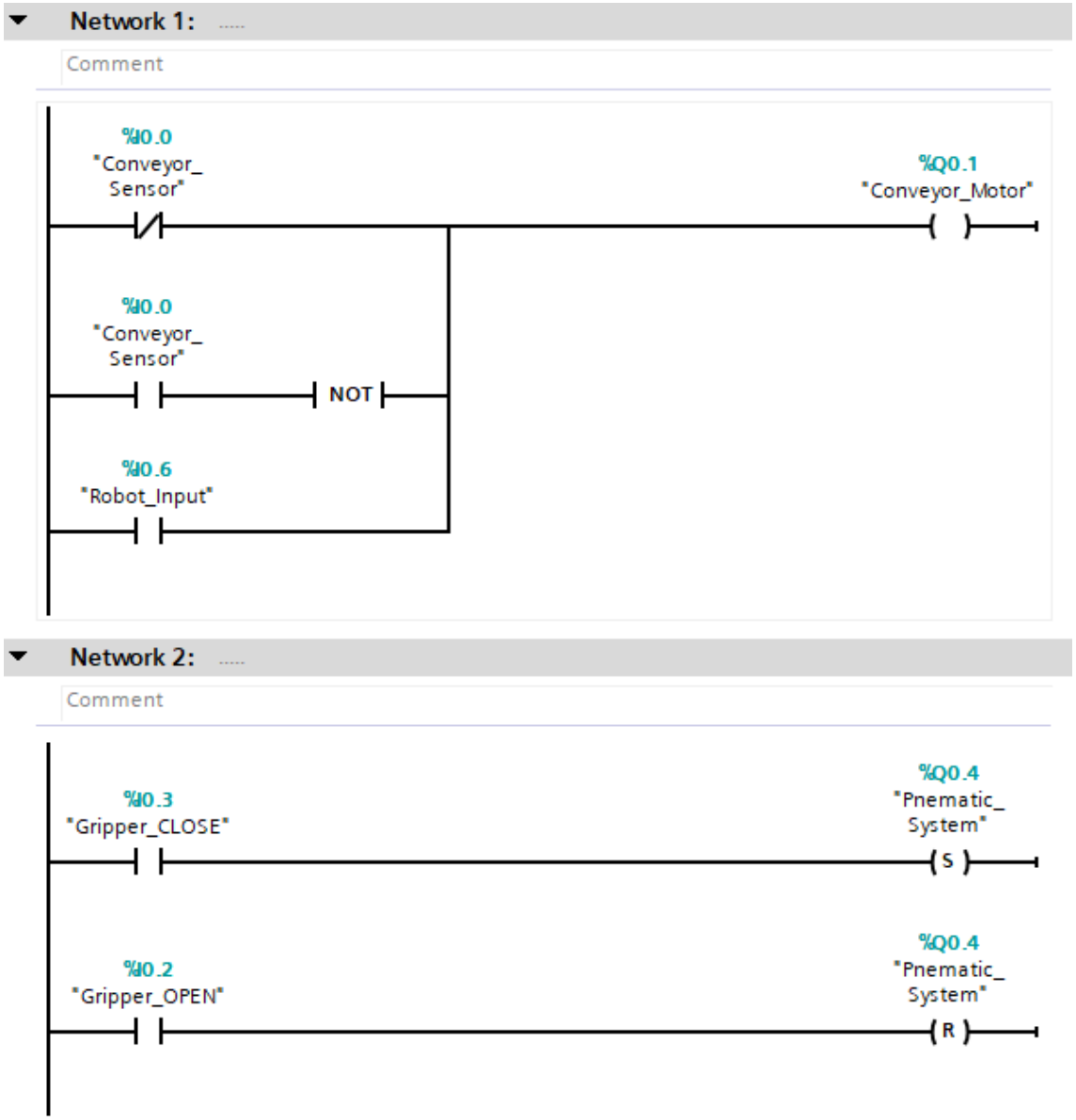


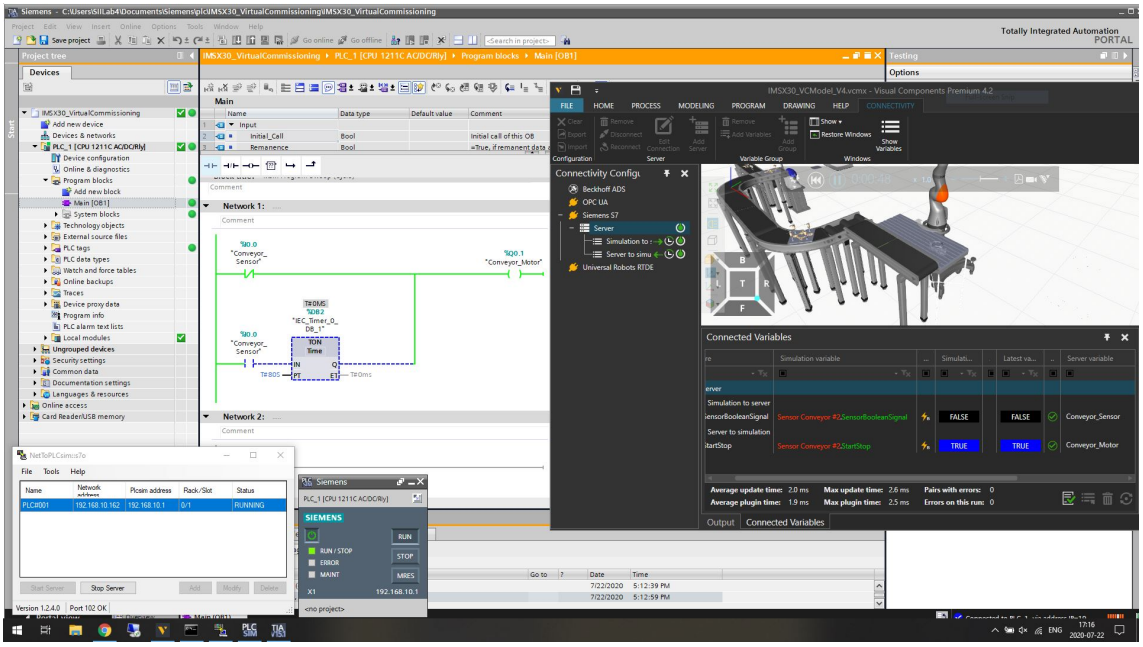

**Figure 4.16:** PLC ladder logic

## 4.7 Virtual commissioning with SiL method

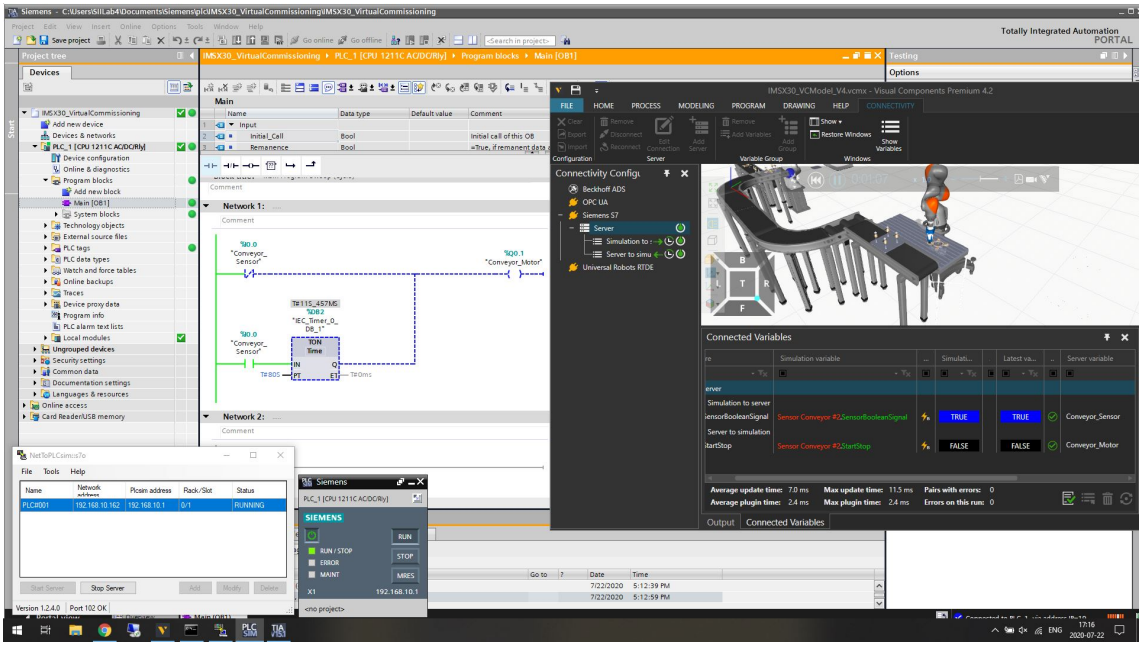
Virtual Commissioning with SiL method is conducted and the results are presented in this section. The process was performed with Visual Components application used in the project. The methodology followed is presented in chapter 3, section 3.7.

### 4.7.1 Visual components

The result obtained from Virtual Commissioning process is presented in figure 4.17. The figure 4.17a represents the first instance when the conveyor motor near KUKA robot station is active. The status is displayed as True in the connected variables tab of Visual Components. However, in the second instance, drone parts are sensed by the sensor and the conveyor motor is stopped as displayed in figure 4.17b. In this stage, sensor is active and displayed as True, whereas conveyor is stopped and displayed as False. The above process aids in testing the programmable logic with simulated environment of drone assembly station.



(a) Conveyor motor - active, sensor - inactive



(b) Sensor - active, conveyor motor - inactive for 80 sec

**Figure 4.17:** SiL with Visual Components

# 5
# Discussion

This section discusses the designed methodology to perform Virtual commissioning and the obtained results, to achieve the objectives and purposes of this master thesis by answering the intended research questions.

With an increase in product variety, manufacturers must be more flexible to produce new products, thus requiring new manufacturing processes to meet the market demands [32]. Hence, the companies adopt new technologies that improve their commissioning processes, to identify bottlenecks and defects before physical implementation. It is possible today to perform commissioning virtually by creating a replica of the real model. Thus, Virtual Commissioning is an approach that creates the replica of a real-world with a real controller to enable complete verification of the manufacturing system [1].

To perform Virtual Commissioning, everything in the real world requires an equivalent in the virtual world. Real-world consists of a real controller, HMI, sensors, actuators, and a machine to perform the tasks [40]. In the same way, a virtual world consists of a virtual controller, HMI Simulation and a virtual station to emulate the real world [40]. The documented procedures, benefits, methods are studied to draft the adopted process methodology. The aforementioned are presented in detail in section 2.6. Virtual Commissioning is widely used in areas like manufacturing, food & beverage and packaging. A unique opportunity of working remotely was explored with this project. All the stages in the adopted methodology can be executed remotely except physical implementation. The importance of data and information exchange within the stages for iterative improvements are realised through this thesis project. This presents a setback in the industrial scenario while implementing Virtual Commissioning, as it demands excessive collaboration between various departments and suppliers.

The selection of a software platform to perform Virtual Commissioning depends on many factors like the cost of the software, availability of an online library, ability to support different Virtual Commissioning methods and expected outcomes of the project. The current major Virtual Commissioning software providers are Siemens, ABB, KUKA, Dassault Systems and Visual Components. Each software application has its own benefits and drawbacks. Hence, it is important to choose the right software based on project requirements. Each software package takes a significant amount of learning time for execution, so it is also important to consider the available time frame for the project.

Modeling of components to build simulation models, occupies a major part of the Virtual Commissioning process. Hence, availability of an online CAD library is an important requirement. Additionally, the drone assembly station is developed to test the performance of the KUKA iiwa robot for drone assembly application using Virtual Commissioning method. The complex electrical system of the drone factory at SII-Lab, proved difficult to separate the already established electrical connections for the assembly station. This urged us to adopt Software in loop (SiL) Virtual Commissioning method to verify the logic behaviour. Most of the available software applications are capable of performing Virtual Commissioning process. However, software applications licensed by Chalmers University are used in this project. Thus, above factors motivated the utilization of Visual Components (VC) and Tecnomatix Process Simulate (TPS) for the project.

The drone assembly process is automated using the KUKA iiwa robot in the virtual world, and the designed automation is verified on the physical station. The project is motivated to test the performance of KUKA iiwa robot for force/snap-fit assemblies. Further, the KUKA robot has not been used on aforementioned applications at SII-Lab. Thus, the project results proved the capability of KUKA iiwa robot on force/snap-fit assemblies, to motivate its deployment on several other applications. The hand-guiding feature on the robot can be utilized excessively in teaching coordinate positions and base calibration for a quicker setup time during physical implementation. The project also describes the detailed procedures of developing simulation models, offline programming, robot calibration and Virtual Commissioning methods. Thus, SII-Lab can be encouraged to implement Virtual Commissioning on the ongoing or subsequent projects using the defined procedures to verify ahead of physical implementation.

The Virtual Commissioning method designed for drone assembly process on Visual Components is very flexible to changes due to the availability of an extensive CAD library. The CAD models in Visual Components are also parametrized, which makes it easier to modify them based on the requirements. The signal connection feature on Visual Components is easy and flexible since the signals for robot, gripper and conveyor are pre-established and only need to be mapped appropriately before testing. Several features, such as drag and drop, Plug n Play feature to move the components, aids in expediting the process.

In case of standalone Tecnomatix Process Simulate, the absence of an online CAD library, leads to investing more time on modeling of components. CAD models of components used are to be imported while building the virtual station. Also, modification of CAD models need to performed externally before updating on Tecnomatix Process Simulate (TPS). However, the main advantage of using TPS, is the availability of frame management option. The feature helps in easy manipulation of objects on the main window, by allowing users to manipulate from one frame to another. The frame management system on TPS, also aids in calibrating the simulation model according to values obtained from physical calibration procedure. The

modification of position and orientation of objects in working space is also easier on TPS due to the availability of path editor panel. The operation sequence can be modified on the sequence editor panel providing flexibility during simulation. Line simulation mode is a robust feature used during Virtual Commissioning.

Force and torque analysis results explained in section 4.5.3, shows that the obtained force/torque values do not vary significantly for both correct and wrong processes. The measured force/torque values also fluctuate with every trial due to low mass of the 3D printed components. Thus, resulting in difficulty to differentiate between the correct and wrong processes. Another factor influencing the minor variation in force/torque values is the gripper orientation during the assembly process. However, considering the involved factors and inconsistencies, it was possible to achieve more than 50% success in identifying correct and wrong processes using measured force/torque values. Modification of gripper fingers, orientation during assembly, increased weight of drone components might lead to improved results. However, the force/torque sensors on KUKA iiwa robot have proven to be beneficial during assembly operations.

# 6
# Conclusion

The aim of the master thesis project is to automate the drone assembly process at Stena industry innovation lab. The assembly process involves snap/force fit assembly of drone frame chassis with drone motors using KUKA LBR iiwa robot. Virtual Commissioning method proved to be ideal to achieve the results by creating a virtual model of the assembly cell and integrating with a virtual PLC. The project is also successful in answering the research questions pertaining to Virtual Commissioning. Both Visual Components (VC) and Tecnomatix Process Simulate (TPS), have proved to be efficient and robust software applications that can be used in Virtual Commissioning.

Virtual Commissioning process has proved to be an important method that can be explored by manufacturing organizations through this project. The process can also aid in better planning and cost saving solutions thus improving the manufacturing capabilities without production stoppage. An interesting opportunity of working remotely that could lead to developments in remotely operated factories and add another dimension to the Industry 4.0 technologies was explored with the thesis project. The project successfully implemented the drone assembly process using the 7-axis KUKA iiwa robot at Stena industry innovation lab.

## 6.1 Future work

The master thesis project also presents topics that can be further explored as future work.

The possibility of adding product weight and physics to the CAD models on simulation platforms can further improve the accuracy and solve practical issues during physical implementation. The possibility of using plugins with simulation applications to incorporate force/torque measurements at the robot TCP could aid in analysis and incorporate ideal solutions to physical models. A possibility of automating the process of manual updation of coordinate points in Sunrise workbench from simulation platforms could be explored. Finally, to link the drone assembly process with the Thingworx system established at Stena industry innovation lab for analysis of data.

# Bibliography


[1] Lee, C. G., & Park, S. C. (2014). *Survey on the virtual commissioning of manufacturing systems.*Journal of Computational Design and Engineering, 1(3), 213–222. https://doi.org/10.7315/jcde.2014.021

[2] Metzner, M., Krieg, L., Merhof, J., Ködel, T., & Franke, J. (2019). *Intuitive Interaction with Virtual Commissioning of Production Systems for Design Validation.* , 892–895. https://doi.org/10.1016/j.procir.2019.08.004

[3] Stena Industry Innovation Lab. (2019, March 22). *The Drone factory in SIILab* [Video file]. Retrieved from https://www.youtube.com/watch?v=86zvGSZfIPM

[4] LBR iiwa. (n.d.). Retrieved February 27, 2020, from https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa

[5] Frank, A. G., Dalenogare, L. S., & Ayala, N. F. (2019). Industry 4.0 technologies: *Implementation patterns in manufacturing companies.* International Journal of Production Economics, 210, 15–26. https://doi.org/10.1016/j.ijpe.2019.01.004

[6] Mattsson, S. (2018a). *Towards increasing operator wellbeing and performance in complex assembly.* Retrieved from https://www.researchgate.net/publication/322504726_Towards_increasing_operator_wellbeing_and_performance_in_complex_assembly

[7] Frohm & Lindström & Winroth & Stahre, J. & V. and M. & J. (2008). Ergonomia - International Journal of Ergonomics and Human Factors. *Levels of Automation in Manufacturing*, 30, 181–207. Retrieved from http://publications.lib.chalmers.se/records/fulltext/76667/local_76667.pdf

[8] Mattsson, S. (2018, September). Level of Automation (LoA) in a production system 14 / 71 [Slides]. Retrieved from https://pingpong.chalmers.se/courseId/9778/node.do?id=4848679&ts=1536228072443&u=341219787

[9] Improve Efficiency With Flexible Automation. (n.d.). Retrieved May 21, 2020, from https://www.crossco.com/resources/technical-bulletins-guides/fixed-versus-flexible-automation/

[10] Halevi, G. (2013). *Process and Operation Planning: Revised Edition of The Principles of Process Planning: A Logical Approach* (Revised ed.). Springer. https://doi.org/10.1007/978-94-017-0259-1

[11] BROWNE, J., TIERNEY, K., & WALSH, M. (1991). A two-stage assembly process planning tooJ for robot-based flexible assembly systems. *International Journal of Production Research, 29(2)*, 247–266. https://doi.org/10.1080/00207549108930068

[12] International Federation of Robotics. (n.d.). Retrieved May 3, 2020, from https://www.ifr.org/industrial-robots/

[13] Kihlman, H. (2019, March 26). Introduction to industrial robotics [Slides]. Retrieved from https://pingpong.chalmers.se/courseId/10677/node.do?id=5255102&ts=1553769449810&u=341219787

[14] KUKA Deutschland GmbH. (2018). *KUKA Sunrise.OS 1.16, KUKA Sunrise.Workbench 1.16 (1.16)* [Operating and Programming Instructions for System Integrators]. Pub KUKA Sunrise.OS 1.16 SI (PDF) en. https://www.kuka.com

[15] GUSTAV BERGSTRÖM. (2011). *Method for calibration of off-line generated robot program*. Gothenburg, Sweden: Chalmers University of Technology.

[16] Per Nyqvist. (2019, April). *Offline programming of an ABB industrial robot*. Gothenburg, Sweden: Chalmers University of Technology.

[17] LEONI Americas. (n.d.). *Robot & tool calibration*. Retrieved July 1, 2020, from https://www.leoni-americas.com/us/products-services/factory-automation-robotics/robot-tool-calibration/

[18] *Robot Calibration*. (n.d.). Retrieved July 1, 2020, from https://robodk.com/doc/en/Robot-Calibration-Creaform.html

[19] Schröer, K. (1998). *Handbook on Robot Performance Testing and Calibration*. Fraunhofer-IRB-Verlag. https://books.google.se/books?id=hNxTtwAACAAJ

[20] Zhang, W., Ma, X., Cui, L., & Chen, Q. (2008). 3 Points Calibration Method of Part Coordinates for Arc Welding Robot. *Intelligent Robotics and Applications*, 216–224. https://doi.org/10.1007/978-3-540-88513-9_24

[21] Bolmsjö, G. (2014). *Programming and simulation of robots*. Retrieved from https://chalmers.instructure.com/courses/9395/files/445917?module_item_id=69837

[22] KUKA Roboter GmbH. (2017). *KUKA Sensitive robotics LBR iiwa*. Retrieved from https://www.kuka.com/-/media/kuka-downloads/imported/9cb8e311bfd744b4b0eab25ca883f6d3/kuka_lbr_iiwa_brochure_en.pdf?rev=5a25f7eac825492e92af6343dbf5bc6b

[23] KUKA Roboter GmbH. (2015). *LBR iiwa Specification*. Retrieved from http://www.oir.caltech.edu/twiki_oir/pub/Palomar/ZTF/KUKARoboticArmMaterial/Spez_LBR_iiwa_en.pdf

[24] Dong, Y., Ren, T., Wu, D., & Chen, K. (2020). *Compliance Control for Robot Manipulation in Contact with a Varied Environment Based on a New Joint Torque Controller*. Journal of Intelligent & Robotic Systems, 99(1), 79–90. https://doi.org/10.1007/s10846-019-01109-8

[25] Bélanger-Barrette, M. (n.d.). Robot Force Torque Sensor - An Introduction. Retrieved September 13, 2020, from https://blog.robotiq.com/bid/72422/Robot-Force-Torque-Sensor-An-Introduction

[26] Vinay Chawda and Gunter Niemeyer. (2017, September). *Toward Torque Control of a KUKA LBR IIWA for Physical Human-Robot Interaction*. Vancouver, BC, Canada: International Conference on Intelligent Robots and Systems.

[27] Bolton, W., & Bolton, W. (2009). *Programmable logic controllers*. Retrieved from https://ebookcentral.proquest.com

[28] Lynn, P. [Udemy]. (2020, April 1). *Learn PLC Programming From Scratch (PLC I)* [Video file]. Retrieved from https://www.udemy.com/course/plc-programming-from-scratch/

[29] Mortensen, S. T., & Madsen, O. (2018). *A Virtual Commissioning Learning Platform*. Procedia Manufacturing, 23, 93–98. https://doi.org/10.1016/j.promfg.2018.03.167

[30] Visual Components. (2016, March 18). *Engineering project with and without virtual commissioning* [Timeline]. What Is Virtual Commissioning? https://www.visualcomponents.com/insights/articles/increasing-control-software-quality-with-virtual-commissioning/

[31] Ko, M., Ahn, E., & Park, S. C. (2013). *A concurrent design methodology of a production system for virtual commissioning*. Concurrent Engineering, 21(2), 129–140. https://doi.org/10.1177/1063293x13476070

[32] Dumitrascu, N.A., Dinca, A., & Predincea, N. (2017). *Virtual commissioning of a robotic cell using Tecnomatix process simulate.* Annals – Series on Engineering, 9(1), 45–60. Retrieved from http://aos.ro/editura/analeleaosr/annals-on-engineering/archive/vol-9-no-1-2017-annals-series-on-engineering-sciences

[33] Shahim, N., & Møller, C. (2016). *Economic justification of Virtual Commissioning in automation industry.* https://doi.org/10.1109/WSC.2016.7822282

[34] KUKA Roboter GmbH. (2016). Media Flange. Retrieved from https://schunk.com/se_en/services/tools-downloads/operating-manuals/list/series/Co-act%20EGP-C/co-act-egp-c

[35] Visual Components. (n.d.). *Connect to a Siemens S7 PLC.* Visual Components Academy. Retrieved July 22, 2020, from https://academy.visualcomponents.com/lessons/connect-to-a-siemens-s7-plc/

[36] Lechler, T., Fischer, E., Metzner, M., Mayr, A., & Franke, J. (2019). Virtual Commissioning – Scientific review and exploratory use cases in advanced production systems. *Procedia CIRP*, 81, 1125–1130. https://doi.org/10.1016/j.procir.2019.03.278

[37] Winther, S. (2017). *Virtual commissioning of production process* (Master Thesis). Chalmers University of Technology. https://odr.chalmers.se/bitstream/20.500.12380/250446/1/250446.pdf

[38] YAO, C., & DZINIC, J. (2013). *Simulation-based verification of PLC programs* (Master Thesis). Chalmers University of Technology. https://odr.chalmers.se/bitstream/20.500.12380/195493/1/195493.pdf

[39] Älegård, S., & Knutsson, S. (2017). *Virtual Commissioning of Smart Factory* (Master Thesis). Chalmers University of Technology. https://odr.chalmers.se/bitstream/20.500.12380/250419/1/250419.pdf

[40] RFID behavior library for SIMIT. (n.d.). Retrieved August 6, 2020, from https://support.industry.siemens.com/cs/document/109766710/rfid-behavior-library-for-simit?dti=&lc=en-WW

[41] Visual Components. (2020, May). *Siemens S7 connection plugin tutorial* https://academy.visualcomponents.com/lessons/connect-to-a-siemens-s7-plc/

# A

# Appendix

## A.1 Procedure for modelling a gripper on Visual Components

1. Import CAD file or add the source of the geometry file into the e-catalogue panel.
2. The source addition option displays all the CAD files in the e-catalogue after importing. However, it is easier to use the import option.
3. The part file of the component is dragged to the 3D world and moved to the world coordinate origin.
4. The component graph is accessed through the modeling tab.
5. The components could be at an offset from the world coordinate origin if the up-axis defined in CATIA V5 could vary from Visual Components.
6. This is fixed by creating a box at the world reference coordinates. Modeling tab -> Geometry -> Features -> Box.
7. The offset distance is measured using the measure option. Modeling tab -> Tools -> Measure
8. The measured values are entered in the component properties tab to remove the offset.
9. The box feature is deleted and gripper parts are selected. Further, right click the mouse button and Collapse features in tools tab is selected. This removes the offset and places the gripper in the world coordinate origin.
10. The features in gripper geometry are extracted and selected features of the component are converted into nodes. This step is performed to create the translational movement of gripper fingers without moving other parts of the gripper.
11. The feature is selected in the component graph panel and extracted into nodes by selecting the explode option.
12. The required geometries selected by holding down ctrl button and right click to select extract -> extract link option. This helps in moving the selected features into a new node and create a link in the component graph panel. This step is repeated for the other gripper finger features.
13. The next step involves defining the DOF and joints of the links.
14. The link properties are selected and joint type is defined as translational. The linear motion is chosen to be along the negative Y axis.
15. For the Schunk Co-act gripper used in the thesis project, following joint properties were defined.

16. Similar process was followed for Link 2, where the joint type was defined as translational follower. The joints are tested by clicking interact button in manipulation group and moving the joints of schunk gripper.
17. To control the actions of gripper during simulation, a servo controller is added in the link properties.
18. Further, the gripper is defined as an end effector for robots by choosing 'end effector' option from wizards tab and selecting IO controls.
19. This option provides a mount frame, tcp and signal options for grasp and release.
20. Tool container option is added to choose the tcp of the gripper
21. Finally, the component is saved and used as a gripper with robots using the pnp option.

## A.2 Procedure for building simulation model and robot programming on Visual Components

1. The layout for the assembly process is built by dragging and dropping the required components onto the main screen.
2. Initially, Work process from works library was dragged on to the main screen. The work process was used to create drone frame and motor components and are transported.
3. The work process was attached to conveyors and a layout was built with a Yaskawa HC10 and KUKA iiwa robot.
4. Yaskawa HC10 robot was used to pick and place the drone components and KUKA iiwa robot was used to conduct the assembly operation.
5. The KUKA robot was selected and opened the programming tab. The sequence file is created and robot programming steps are created.
6. Robot coordinates are created easily on Visual components by choosing the snap function and selecting the product to be picked.
7. This option displays the gripper and robot position at the product.
8. A coordinate point is created by choosing the PTP or LIN movement at the snapped position and thus the robot movement is programmed.
9. The gripper operation of grasping and releasing is set by setting input and output signals to true or false.
10. The signal numbers for the Schunk gripper used in the project are 101 and 102 for grasp and release operations.

# B

# Appendix

## B.1 Procedure for setting kinematics on Tecnomatix process simulate

1. The .jt file extension is imported into the software application.
2. Resource or part option is chosen during importing of .jt files to .cojt format.
3. KUKA iiwa robot and Schunk gripper are imported as resources. Drone frame, motors, pallet are imported as parts.
4. KUKA iiwa robot is selected from the object tree in resources tab and modelling tab is opened.
5. Set modeling scope option is chosen.
6. Kinematics editor is opened. Links are created by choosing, Create link option.
7. KUKA iiwa robot consists of eight links. Each link is associated to part in the link properties panel.
8. Joints between the links are created by choosing create joints option or by connecting the links with an arrow head.
9. Joint properties including joint name, axis points, joint type i.e. revolute, joint axis limits, joint speed and acceleration details are filled.
10. similar process is followed for the remaining links and joints.
11. Baseframe for the robot is set by choosing 'set baseframe' option in the kinematics editor and the frame coordinates are chosen on link 1.
12. The details mentioned above are extracted from the KUKA iiwa robot manual that are explained in the chapter 2, table 2.3b.
13. Similar process is followed for setting toolframe of the robot. The toolframe of the robot is also known as flange coordinates.
14. Inverse kinematics for more than six degrees of freedom are automatically set while exiting the kinematics editor.
15. End modeling scope option is chosen to save the work.
16. Similar process is followed for Schunk gripper model, where baseframe and TCP are set along with joint types as translational and translation follower instead of revolute joint as set for KUKA robot.

## B.2 Procedure for assembly process simulation on Tecnomatix process simulate

1. Operation tab at the top of the window is selected to simulate the assembly process.
2. The processes created are displayed in the operation tree at the bottom left corner of the window.
3. The first operation was the flow of drone components on the pallet.
4. The pallet along with drone components flows on the conveyor and stops at the defined point.
5. New object flow operation is created and name of the operation is filled eg. Drone frame flow.
6. Object is selected as drone frame either from the simulation window or from the object tree.
7. start and end point coordinated are chosen on the simulation window.
8. Similar process is followed for drone motors and pallet.
9. The operations are added within a compounded operation and named are drone comp flow.
10. The robot coordinates for movement are provided through generic robotic operation.
11. The robot is chosen that automatically chooses the tool mounted in the robot for the operation.
12. Add current location is selected that provides the first coordinate point of the robot.
13. Further points for robot movement are added by selecting 'add location after' option.
14. The window with coordinate points, joint angles are displayed that can be used to set the next location.
15. TCP coordinates can also be dragged or manipulated that moves the robot with respect to defined kinematics to the desired location.
16. Teach location option is selected after the desired point is chosen.
17. The process is repeated to define coordinate points that are to be followed by the robot.
18. After reaching the required location, Gripper operation is created.
19. The gripper in use is selected in the gripper operation window. The action required to be performed by the gripper is chosen along with the frame i.e. TCP.
20. Thus the robotic operation steps are repeated for the assembly operation.
21. A compounded operation is created for the tasks such as drone frame assembly, motor assembly are created and the respective tasks or points are dragged and dropped in the compounded operation.
22. Finally, the complete operation is dragged onto the sequence editor where, a gantt chart of the tasks is created.
23. The order of the tasks to be performed are set in the gantt chart by connecting subsequent task through arrow heads.

24. The operation is dragged and dropped onto the Path editor for obtaining the coordinates with respect to the working frame and also setting of OLP commands.
25. OLP commands are set at coordinate points for providing simulation commands such as Attach components, detach components, check and release grippers etc.

# C
# Appendix

## C.1 Procedure for tool calibration

### C.1.1 XYZ 4-point method [14]

**Precondition**

1. The tool to be calibrated is mounted on the mounting flange.
2. The tool to be calibrated and the frame used as the TCP have been created in the object templates of the project and transferred to the robot controller by means of synchronization.
3. T1 mode.

**Procedure**

1. Select Calibration > Tool calibration at the Robot level. The Tool calibration view opens.
2. Select the tool to be calibrated and the corresponding TCP.
3. Select the TCP calibration (XYZ 4-point) method. The measuring points of the method are displayed as buttons: Measurement point 1 ... Measurement point 4. In order to be able to record a measuring point, it must be selected (button is orange).
4. Move the TCP to any reference point. Press Record calibration point. The position data are applied and displayed for the selected measuring point.
5. Move the TCP to the reference point from a different direction. Press Record calibration point. The position data are applied and displayed for the selected measuring point.
6. Repeat step 5 two more times.
7. Press Determine tool data. The calibration data and the calculation error are displayed in the Apply tool data dialog.
8. If the calculation error exceeds the maximum permissible value, a warning is displayed. Press Cancel and recalibrate the TCP.
9. If the calculation error is below the configured limit, press Apply to save the calibration data.
10. Close the Calibration view or define the orientation of the tool coordinate system with the ABC 2-point.
11. Synchronize the project in order to save the calibration data in Sunrise Workbench.

### C.1.2 ABC 2-point method [14]

**Precondition**

1. The tool to be calibrated is not a safety-oriented tool.
2. The tool to be calibrated is mounted on the mounting flange.
3. The TCP of the tool has already been measured.
4. T1 mode.

**Procedure**

1. Only if the Calibration view was closed following TCP calibration: Select Calibration > Tool calibration at the Robot level. The Tool calibration view opens.
2. Only if the Calibration view was closed following TCP calibration: Select the mounted tool and the corresponding TCP of the tool.
3. Select the Defining the orientation(ABC 2-point) method. The measuring points of the method are displayed as buttons: • TCP • Negative X axis • Positive Y value on XY plane. In order to be able to record a measuring point, it must be selected (button is orange).
4. Move the TCP to any reference point. Press Record calibration point. The position data are applied and displayed for the selected measuring point.
5. Move the tool so that the reference point on the X axis has a negative X value (i.e. move against the tool direction). Press Record calibration point. The position data are applied and displayed for the selected measuring point.
6. Move the tool so that the reference point in the XY plane has a positive Y value. Press Record calibration point. The position data are applied and displayed for the selected measuring point.
7. Press Determine tool data. The calibration data are displayed in the Apply tool data dialog.
8. Press Apply to save the calibration data.
9. Synchronize the project in order to save the calibration data in Sunrise Workbench.

## C.2 Procedure for base calibration

### C.2.1 3-point method [14]

**Precondition**

1. A previously calibrated tool is mounted on the mounting flange.
2. The frame to be calibrated has been selected as the base in the application data of the project and transferred to the robot controller by means of synchronization.
3. T1 mode.

**Procedure**

1. Select Calibration > Base calibration at the Robot level. The Base calibration view opens.
2. Select the base to be calibrated.

3. Select the mounted tool and the TCP of the tool with which the measuring points of the base are addressed. The measuring points of the 3-point method are displayed as buttons: • Origin • Positive X axis • Positive Y value on XY plane. In order to be able to record a measuring point, it must be selected (button is orange).
4. Move the TCP to the origin of the base. Press Record calibration point. The position data are applied and displayed for the selected measuring point.
5. Move the TCP to a point on the positive X axis of the base. Press Record calibration point. The position data are applied and displayed for the selected measuring point.
6. Move the TCP to a point in the XY plane with a positive Y value. Press Record calibration point. The position data are applied and displayed for the selected measuring point.
7. Press Determine base data. The calibration data are displayed in the Apply base data dialog.
8. Press Apply to save the calibration data.
9. Synchronize the project in order to save the calibration data in Sunrise Workbench.

## C.3 Variation analysis results

**Table C.1:** Variation analysis results

| **Base Calibration Results from Variation Analysis Experiment** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Positions** | **Hand-Guiding Method (HM)** | | | **3-Point Method (3PM)** | | | **Difference = (3PM - HM)** | | |
| | **Coordinate Points** | | | | | | | | |
| | **X** | **Y** | **Z** | **X** | **Y** | **Z** | **X** | **Y** | **Z** |
| 1 | 125.54 | 612.87 | -102.91 | 127.75 | 611.1 | -101.4 | 2.21 | -1.77 | 1.51 |
| 2 | 139.54 | 612.35 | -103.29 | 140.71 | 613.14 | -102.27 | 1.17 | 0.79 | 1.02 |
| 3 | 170.18 | 559.91 | -98.99 | 171.68 | 560.91 | -97.88 | 1.5 | 1 | 1.11 |
| 4 | 231.27 | 579.94 | -101.06 | 230.36 | 578.3 | -99.24 | -0.91 | -1.64 | 1.82 |
| 5 | 263.65 | 508.98 | -101.19 | 264.03 | 508.74 | -100.58 | 0.38 | -0.24 | 0.61 |
| 6 | 237.52 | 623.69 | -103.53 | 236.86 | 622.57 | -103.29 | -0.66 | -1.12 | 0.24 |
| 7 | 118.87 | 558.42 | -106.57 | 118.84 | 557.53 | -106.38 | -0.03 | -0.89 | 0.19 |
| 8 | 150.57 | 519.1 | -102.6 | 150.97 | 517.53 | -102.38 | 0.4 | -1.57 | 0.22 |
| 9 | 248.07 | 514.92 | -103.15 | 250.84 | 515.04 | -102.8 | 2.77 | 0.12 | 0.35 |
| 10 | 120.04 | 601.87 | -100.51 | 122.35 | 602.87 | -100.67 | 2.31 | 1 | -0.16 |
| 11 | 237.07 | 546.6 | -103.47 | 235.89 | 543.97 | -102.3 | -1.18 | -2.63 | 1.17 |
| 12 | 230.62 | 590.44 | -105.19 | 231.52 | 589.91 | -104.44 | 0.9 | -0.53 | 0.75 |
| 13 | 213.05 | 519.39 | -107.01 | 211.27 | 516.2 | -107.2 | -1.78 | -3.19 | -0.19 |

# D
# Appendix

## D.1 Coordinate points

**Table D.1:** Coordinate points retrieved from Tecnomatix Process Simulate

| Paths & Locations | Type | X | Y | Z | RX | RY | RZ | Duration |
|---|---|---|---|---|---|---|---|---|
| Drone Assembly Operation | CompoundOperation | | | | | | | 109.531 |
| Generate Drone Parts | Task | | | | | | | 0 |
| Drone Comp Flow | CompoundOperation | | | | | | | 5 |
| Pallet Flow | PmObjectFlowOperation | | | | | | | 5 |
| loc2 | PmObjectFlowLocationOperation | 618.5 | 1343.9 | -43.2 | 180 | 0 | 0 | 0 |
| loc3 | PmObjectFlowLocationOperation | 624.1 | 540.7 | -43.2 | 180 | 0 | 0 | 5 |
| Drone Frame Flow | PmObjectFlowOperation | | | | | | | 5 |
| loc4 | PmObjectFlowLocationOperation | 618.5 | 1343.9 | -80.5 | 180 | 0 | 45 | 0 |
| loc5 | PmObjectFlowLocationOperation | 624.1 | 540.8 | -80.5 | 180 | 0 | 45 | 5 |
| Motor 3 Flow | PmObjectFlowOperation | | | | | | | 5 |
| loc6 | PmObjectFlowLocationOperation | 540.4 | 1420.9 | -180.3 | -180 | 0 | 180 | 0 |
| loc7 | PmObjectFlowLocationOperation | 540.4 | 618.1 | -180.3 | -180 | 0 | 180 | 5 |
| Motor 1 Flow | PmObjectFlowOperation | | | | | | | 5 |
| loc8 | PmObjectFlowLocationOperation | 695.7 | 1420.9 | -180.3 | -180 | 0 | 180 | 0 |
| loc9 | PmObjectFlowLocationOperation | 695.7 | 618.1 | -180.3 | -180 | 0 | 180 | 5 |
| Motor 2 Flow | PmObjectFlowOperation | | | | | | | 5 |
| loc10 | PmObjectFlowLocationOperation | 540.4 | 1265.4 | -180.3 | -180 | 0 | 180 | 0 |
| loc11 | PmObjectFlowLocationOperation | 540.4 | 462.6 | -180.3 | -180 | 0 | 180 | 5 |
| Motor 4 Flow | PmObjectFlowOperation | | | | | | | 5 |
| loc12 | PmObjectFlowLocationOperation | 695.3 | 1265.3 | -180.3 | -180 | 0 | 180 | 0 |
| loc13 | PmObjectFlowLocationOperation | 695.3 | 462.6 | -180.3 | -180 | 0 | 180 | 5 |
| Drone Frame Assembly | CompoundOperation | | | | | | | 21.985 |
| Gripper Open | Task | | | | | | | 0.126 |
| Move to Pt | PmGenericRoboticOperation | | | | | | | 2.39 |
| via1 | PmViaLocationOperation | 540 | 541 | -186.5 | 0 | 0 | -90 | 1.749 |
| via2 | PmViaLocationOperation | 540 | 541 | -84 | 0 | 0 | -90 | 0.64 |
| Grasp Drone Frame | Task | | | | | | | 0.126 |
| Assemble Drone Frame | PmGenericRoboticOperation | | | | | | | 16.43 |
| via3 | PmViaLocationOperation | 540 | 541 | -84 | 0 | 0 | -90 | 0 |
| via4 | PmViaLocationOperation | 540 | 541 | -234 | 0 | 0 | -90 | 0.775 |
| via5 | PmViaLocationOperation | 337.9 | 548.3 | -380.7 | 0 | 0 | -90 | 1.206 |
| via6 | PmViaLocationOperation | 325 | 258.8 | -176 | 0 | 0 | 135.1 | 1.996 |
| via8 | PmViaLocationOperation | 325 | 258.8 | -98.5 | 0 | 0 | 135.1 | 12.454 |
| Release Drone Frame | Task | | | | | | | 0.126 |
| Motor Assembly | CompoundOperation | | | | | | | 59.734 |
| Motor 1 Assembly | PmGenericRoboticOperation | | | | | | | 6.284 |
| via9 | PmViaLocationOperation | 325.9 | 259.2 | -98.5 | 0 | 0 | 134.6 | 0.149 |
| via10 | PmViaLocationOperation | 326 | 260.6 | -308 | 0 | 0 | 134.6 | 2.195 |
| via11 | PmViaLocationOperation | 695.9 | 617.5 | -339.2 | -30 | 0 | -90 | 1.948 |
| via12 | PmViaLocationOperation | 695.9 | 617.5 | -150 | -30 | 0 | -90 | 1.992 |
| Grasp Motor 1 | Task | | | | | | | 0.126 |
| Assemble Motor 1 | PmGenericRoboticOperation | | | | | | | 6.75 |
| via13 | PmViaLocationOperation | 695.9 | 617.5 | -150 | -30 | 0 | -90 | 0.028 |
| via14 | PmViaLocationOperation | 695.9 | 617.5 | -340 | -30 | 0 | -90 | 2 |
| via15 | PmViaLocationOperation | 166.4 | 99.9 | -246.7 | -30 | 0 | -72.9 | 2.24 |
| via16 | PmViaLocationOperation | 165.1 | 99.9 | -123.6 | -30 | 0 | -72.9 | 1.331 |
| via17 | PmViaLocationOperation | 186.3 | 120.7 | -123.6 | -30 | 0 | -72.9 | 0.398 |
| via18 | PmViaLocationOperation | 195.5 | 130.1 | -123.6 | -30 | 0 | -41.1 | 0.753 |
| Release Motor 1 | Task | | | | | | | 0.126 |
| Motor 2 Assembly | PmGenericRoboticOperation | | | | | | | 6.1 |
| via19 | PmViaLocationOperation | 195.5 | 130.1 | -123.6 | -30 | 0 | -41.1 | 0 |
| via20 | PmViaLocationOperation | 195.5 | 130.1 | -327.1 | -0.7 | 0 | -36.5 | 2.135 |
| via44 | PmViaLocationOperation | 190.9 | 130.5 | -327.1 | 0 | 0 | -90 | 1.501 |
| via21 | PmViaLocationOperation | 540.3 | 462.3 | -244.6 | 0 | 0 | -90 | 1.418 |
| via22 | PmViaLocationOperation | 540.3 | 462.3 | -150 | 0 | 0 | -90 | 1.046 |
| Grasp Motor 2 | Task | | | | | | | 0.126 |
| Assemble Motor 2 | PmGenericRoboticOperation | | | | | | | 4.037 |
| via23 | PmViaLocationOperation | 540.3 | 462.3 | -150 | 0 | 0 | -90 | 0 |
| via24 | PmViaLocationOperation | 540.3 | 462.3 | -244.6 | 0 | 0 | -90 | 1.046 |
| via74 | PmViaLocationOperation | 361.5 | 99.5 | -211.3 | 0 | 0 | -90 | 1.236 |
| via25 | PmViaLocationOperation | 361.5 | 99.5 | -123.6 | 0 | 0 | -90 | 0.976 |
| via26 | PmViaLocationOperation | 347.4 | 116.2 | -123.6 | 0 | 0 | -90 | 0.319 |

**Table D.1 continued from previous page**

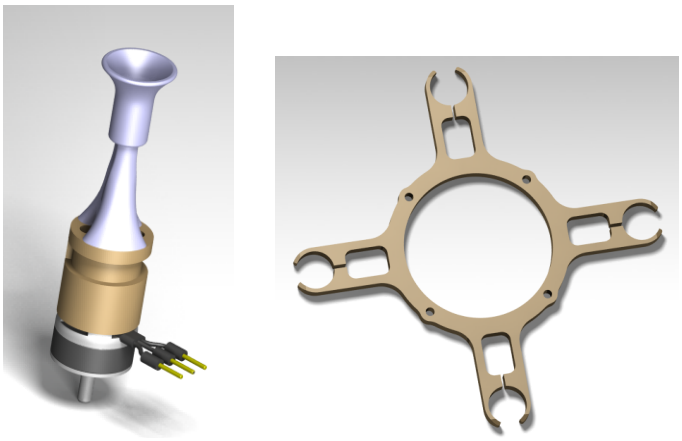
| Paths & Locations | Type | X | Y | Z | RX | RY | RZ | Duration |
|---|---|---|---|---|---|---|---|---|
| via7 | PmViaLocationOperation | 344.4 | 120.2 | -123.6 | 0 | 0 | -90 | 0.235 |
| via27 | PmViaLocationOperation | 336.7 | 128.7 | -123.6 | 0 | 0 | -135 | 0.225 |
| Release Motor 2 | Task | | | | | | | 0.126 |
| Motor 3 Assembly | PmGenericRoboticOperation | | | | | | | 4.392 |
| via45 | PmViaLocationOperation | 336.7 | 128.7 | -123.6 | 0 | 0 | -135 | 0 |
| via46 | PmViaLocationOperation | 336.7 | 128.7 | -280 | 0 | 0 | -135 | 1.664 |
| via47 | PmViaLocationOperation | 539.7 | 616.9 | -280.1 | 0 | 0 | -90 | 1.327 |
| via48 | PmViaLocationOperation | 539.7 | 616.9 | -150 | 0 | 0 | -90 | 1.401 |
| Grasp Motor 3 | Task | | | | | | | 0.126 |
| Assemble Motor 3 | PmGenericRoboticOperation | | | | | | | 6.082 |
| via49 | PmViaLocationOperation | 539.7 | 616.9 | -150 | 0 | 0 | -90 | 0.027 |
| via50 | PmViaLocationOperation | 539.7 | 616.9 | -252.1 | 0 | 0 | -90 | 1.121 |
| via | PmViaLocationOperation | 326.4 | 456 | -252.1 | 0 | 0 | -140 | 1.275 |
| via53 | PmViaLocationOperation | 172.9 | 299.7 | -162.2 | 0 | 0 | -140 | 2.468 |
| via54 | PmViaLocationOperation | 173.4 | 295.4 | -123.6 | 0 | 0 | -170 | 0.489 |
| via55 | PmViaLocationOperation | 181 | 284.4 | -123.6 | 0 | 0 | -170 | 0.234 |
| via41 | PmViaLocationOperation | 186.3 | 278 | -123.6 | 0 | 0 | -170 | 0.246 |
| via56 | PmViaLocationOperation | 193.9 | 269.8 | -123.6 | 0 | 0 | -135 | 0.222 |
| Release Motor 3 | Task | | | | | | | 0.126 |
| Motor 4 Assembly | PmGenericRoboticOperation | | | | | | | 3.418 |
| via57 | PmViaLocationOperation | 193.9 | 269.8 | -123.6 | 0 | 0 | -135 | 0 |
| via58 | PmViaLocationOperation | 193.9 | 269.8 | -310.1 | 0 | 0 | -135 | 1.066 |
| via59 | PmViaLocationOperation | 694.8 | 463 | -252.1 | -30 | 0 | -90 | 1.231 |
| via60 | PmViaLocationOperation | 694.8 | 463 | -150 | -30 | 0 | -90 | 1.121 |
| Grasp Motor 4 | Task | | | | | | | 0.126 |
| ReOrient Motor 4 | PmGenericRoboticOperation | | | | | | | 3.364 |
| via61 | PmViaLocationOperation | 694.8 | 463 | -150 | -30 | 0 | -90 | 0.032 |
| via62 | PmViaLocationOperation | 694.8 | 463 | -252.1 | -30 | 0 | -90 | 1.121 |
| via29 | PmViaLocationOperation | 548.7 | 463 | -252.1 | -30 | 0 | -90 | 1.091 |
| via30 | PmViaLocationOperation | 548.7 | 463 | -150 | -30 | 0 | -90 | 1.121 |
| Release Motor 4 | Task | | | | | | | 0.126 |
| RePick Motor 4 | PmGenericRoboticOperation | | | | | | | 2.321 |
| via28 | PmViaLocationOperation | 548.7 | 463 | -150 | -30 | 0 | -90 | 0.047 |
| via32 | PmViaLocationOperation | 548.7 | 463 | -252.1 | -30 | 0 | -90 | 1.121 |
| via31 | PmViaLocationOperation | 548.4 | 462.9 | -252.1 | 0 | 0 | -90 | 0.033 |
| via33 | PmViaLocationOperation | 548.4 | 462.9 | -150 | 0 | 0 | -90 | 1.121 |
| Grasp Motor 4Of2 | Task | | | | | | | 0.126 |
| Assemble Motor 4 | PmGenericRoboticOperation | | | | | | | 6.879 |
| via34 | PmViaLocationOperation | 548.4 | 462.9 | -150 | 0 | 0 | -90 | 0 |
| via35 | PmViaLocationOperation | 548.4 | 462.9 | -337.1 | 0 | 0 | -90 | 1.971 |
| via36 | PmViaLocationOperation | 387.9 | 319.9 | -337.8 | 0 | 0 | 174 | 1.802 |
| via38 | PmViaLocationOperation | 387.9 | 313.9 | -123.6 | 0 | 0 | 174 | 2.243 |
| via39 | PmViaLocationOperation | 351.5 | 282.8 | -123.6 | 0 | 0 | 174 | 0.579 |
| via37 | PmViaLocationOperation | 338.7 | 271 | -123.6 | 0 | 0 | 135 | 0.285 |
| Release Motor 4Of2 | Task | | | | | | | 0.126 |
| Pick Place Assembled Drone | CompoundOperation | | | | | | | 22.76 |
| Pick Assembled Drone | PmGenericRoboticOperation | | | | | | | 0.185 |
| via67 | PmViaLocationOperation | 338.7 | 271 | -123.6 | 0 | 0 | 135 | 0 |
| via70 | PmViaLocationOperation | 338.7 | 271 | -115 | 0 | 0 | 135 | 0.185 |
| Grasp Assembled Drone | Task | | | | | | | 0.126 |
| Place Assembled Drone | PmGenericRoboticOperation | | | | | | | 7.89 |
| via71 | PmViaLocationOperation | 338.7 | 271 | -115 | 0 | 0 | 135 | 0 |
| via73 | PmViaLocationOperation | 338.7 | 271 | -413.7 | 0 | 0 | 135 | 3.087 |
| via75 | PmViaLocationOperation | -102 | 1242.7 | -407.2 | 0 | 0 | -59.7 | 2.042 |
| via76 | PmViaLocationOperation | -99.7 | 1242.7 | -141 | 0 | 0 | 0 | 2.762 |
| Release Assembled Drone | Task | | | | | | | 0.126 |
| Robot Home | Task | | | | | | | 5 |
| Robot Home | PmGenericRoboticOperation | | | | | | | 4.433 |
| via77 | PmViaLocationOperation | -99.7 | 1242.7 | -141 | 0 | 0 | 0 | 0 |
| via78 | PmViaLocationOperation | -101.7 | 1242.7 | -379.2 | 0 | 0 | 0 | 2.482 |
| via40 | PmViaLocationOperation | 302.7 | 273.9 | -413.7 | 0 | 0 | 150 | 1.95 |

# E

# Appendix

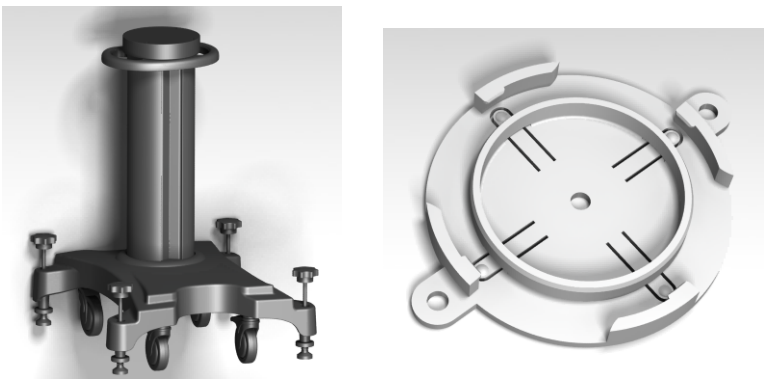
## E.1 Technical data

### E.1.1 CAD model renderings



**(a)** Drone motor **(b)** Drone frame

**Figure E.1:** Drone components

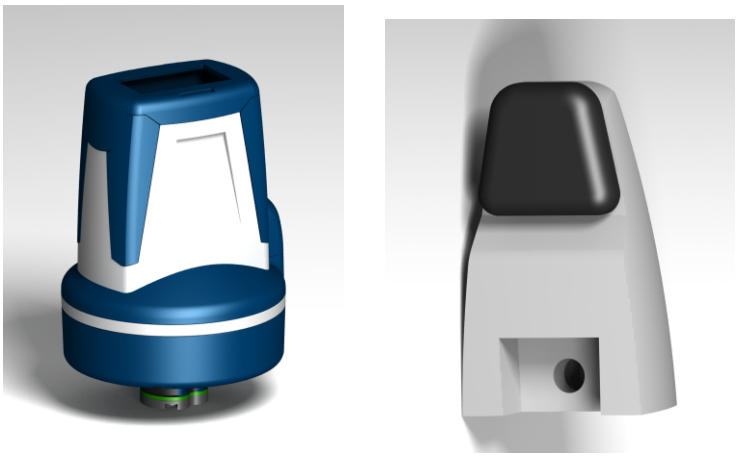


**(a)** Robot pedestal **(b)** Fixture

**Figure E.2:** station components

## E.2 Modeling

### E.2.1 CAD assembly renderings



(a) Schunk gripper part (b) Gripper fingers

**Figure E.3:** Schunk gripper CAD parts

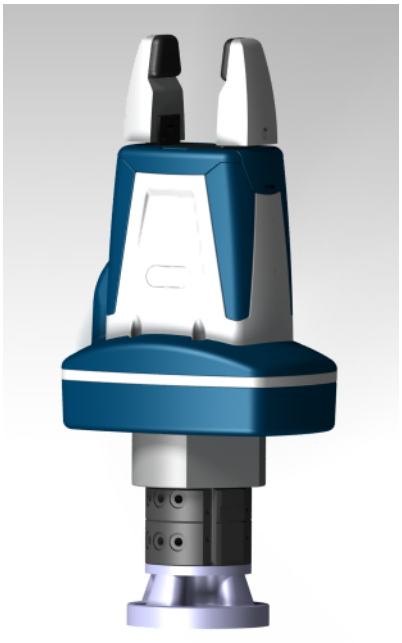


**Figure E.4:** Schunk gripper CAD assembly

# F

# Appendix

## F.1 Robot programming

### F.1.1 Source code - robot assembly

```
package application;

import javax.inject.Inject;
import javax.inject.Named;

import com.kuka.generated.ioAccess.MediaFlangeIOGroup;
import com.kuka.roboticsAPI.applicationModel.RoboticsAPIApplication;
import static com.kuka.roboticsAPI.motionModel.BasicMotions.*;

import com.kuka.roboticsAPI.deviceModel.LBR;
import com.kuka.roboticsAPI.geometricModel.Tool;


/**
 * Implementation of a robot application.
 * <p>
 * The application provides a {@link RoboticsAPITask\#initialize()} and a
 * {@link RoboticsAPITask\#run()} method, which will be called successively in
 * the application lifecycle. The application will terminate automatically after
 * the {@link RoboticsAPITask\#run()} method has finished or after stopping the
 * task. The {@link RoboticsAPITask\#dispose()} method will be called, even if an
 * exception is thrown during initialization or run.
 * <p>
 * <b>It is imperative to call <code>super.dispose()</code> when overriding the
 * {@link RoboticsAPITask#dispose()} method.</b>
 *
 * @see UseRoboticsAPIContext
 * @see \#initialize()
 * @see \#run()
 * @see \#dispose()
 */

public class KUKAIMSX30 extends RoboticsAPIApplication {
        @Inject
        private LBR lBR_iiwa_14_R820_1;
        private MediaFlangeIOGroup MFIO = null;

        @Inject
        @Named("Gripper")
        private Tool Gripper;

        @Override
        public void initialize() {
                // initialize your application here
                Gripper.attachTo(lBR_iiwa_14_R820_1.getFlange());

                if(MFIO == null) {
                        MFIO = new MediaFlangeIOGroup(lBR_iiwa_14_R820_1.getController());
                }
```

```
}

@Override
public void run() {
        // your application execution starts here
        lBR_iiwa_14_R820_1.move(ptpHome());

        droneFrameAssembly();
        motorOneAssembly();
        motorTwoAssembly();
        motorThreeAssembly();
        motorFourAssembly();
        pkplAssembledDrone();

        lBR_iiwa_14_R820_1.move(ptpHome());
        MFIO.setLEDRed(false);
}


private void droneFrameAssembly() {

        //gripper open
        MFIO.setGripperClose(false);
        MFIO.setGripperOpen(true);

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via1")));
        Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/via2")));

        //gripper grasp frame
        MFIO.setGripperOpen(false);
        MFIO.setGripperClose(true);

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via3")));
        Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/via4")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via5")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via6")));
        Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/via7")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via8")));

        //gripper release frame
        MFIO.setGripperClose(false);
        MFIO.setGripperOpen(true);

}

private void motorOneAssembly() {
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via9")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via10")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via11")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via12")));

        //gripper grasp motor1
        MFIO.setGripperOpen(false);
        MFIO.setGripperClose(true);

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via13")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via14")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via15")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via16")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via17")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via18")));

        //gripper release motor1
        MFIO.setGripperClose(false);
        MFIO.setGripperOpen(true);

}

private void motorTwoAssembly() {
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via19")));
```

```
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via20")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via44")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via21")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via22")));

        //gripper grasp motor2
        MFIO.setGripperOpen(false);
        MFIO.setGripperClose(true);

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via23")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via24")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via74")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via25")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via26")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via27")));

        //gripper release motor2
        MFIO.setGripperClose(false);
        MFIO.setGripperOpen(true);
}

private void motorThreeAssembly() {
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via45")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via46")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via47")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via48")));

        //gripper grasp motor3
        MFIO.setGripperOpen(false);
        MFIO.setGripperClose(true);


        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via49")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via50")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via51")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via52")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via53")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via54")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via55")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via56")));

        //gripper release motor3
        MFIO.setGripperClose(false);
        MFIO.setGripperOpen(true);
}

private void motorFourAssembly() {
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via57")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via58")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via59")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via60")));

        //gripper grasp motor4
        MFIO.setGripperOpen(false);
        MFIO.setGripperClose(true);


        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via61")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via62")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via63")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via66")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via64")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via65")));

        //gripper release motor4
        MFIO.setGripperClose(false);
        MFIO.setGripperOpen(true);
}

private void pkplAssembledDrone() {
```

```
                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via67")));
                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via68")));
                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via69")));
                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via70")));

                //gripper grasp drone
                MFIO.setGripperOpen(false);
                MFIO.setGripperClose(true);


                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via71")));
                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via72")));
                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via73")));
                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via75")));
                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via76")));

                //gripper release drone
                MFIO.setGripperClose(false);
                MFIO.setGripperOpen(true);

                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via77")));
                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/via78")));
        }


}
```

### F.1.2 Source code - MFIO flange

```
package com.kuka.generated.ioAccess;

import javax.inject.Inject;
import javax.inject.Singleton;

import com.kuka.roboticsAPI.controllerModel.Controller;
import com.kuka.roboticsAPI.ioModel.AbstractIOGroup;
import com.kuka.roboticsAPI.ioModel.IOTypes;

/**
 * Automatically generated class to abstract I/O access to I/O group <b>MediaFlange</b>.<br>
 * <i>Please, do not modify!</i>
 * <p>
 * <b>I/O group description:</b><br>
 * This I/O Group contains the In-/Outputs for the Media-Flange Touch.
 */
@Singleton
public class MediaFlangeIOGroup extends AbstractIOGroup
{
        /**
         * Constructor to create an instance of class 'MediaFlange'.<br>
         * <i>This constructor is automatically generated. Please, do not modify!</i>
         *
         * @param controller
         *            the controller, which has access to the I/O group 'MediaFlange'
         */
        @Inject
        public MediaFlangeIOGroup(Controller controller)
        {
                super(controller, "MediaFlange");

                addInput("InputX3Pin3", IOTypes.BOOLEAN, 1);
                addInput("InputX3Pin4", IOTypes.BOOLEAN, 1);
                addInput("InputX3Pin10", IOTypes.BOOLEAN, 1);
                addInput("InputX3Pin13", IOTypes.BOOLEAN, 1);
                addInput("InputX3Pin16", IOTypes.BOOLEAN, 1);
                addInput("UserButton", IOTypes.BOOLEAN, 1);
                addDigitalOutput("LEDBlue", IOTypes.BOOLEAN, 1);
                addDigitalOutput("SwitchOffX3Voltage", IOTypes.BOOLEAN, 1);
                addDigitalOutput("ChangerOpen", IOTypes.BOOLEAN, 1);
```

```
        addDigitalOutput("ChangerClose", IOTypes.BOOLEAN, 1);
        addDigitalOutput("LEDGreen", IOTypes.BOOLEAN, 1);
        addDigitalOutput("LEDRed", IOTypes.BOOLEAN, 1);
        addDigitalOutput("GripperClose", IOTypes.BOOLEAN, 1);
        addDigitalOutput("GripperOpen", IOTypes.BOOLEAN, 1);
}

/**
 * Gets the value of the <b>digital input '<i>InputX3Pin3</i>'</b>.<br>
 * <i>This method is automatically generated. Please, do not modify!</i>
 * <p>
 * <b>I/O direction and type:</b><br>
 * digital input
 * <p>
 * <b>User description of the I/O:</b><br>
 * ./.
 * <p>
 * <b>Range of the I/O value:</b><br>
 * [false; true]
 *
 * @return current value of the digital input 'InputX3Pin3'
 */
public boolean getInputX3Pin3()
{
        return getBooleanIOValue("InputX3Pin3", false);
}

/**
 * Gets the value of the <b>digital input '<i>InputX3Pin4</i>'</b>.<br>
 * <i>This method is automatically generated. Please, do not modify!</i>
 * <p>
 * <b>I/O direction and type:</b><br>
 * digital input
 * <p>
 * <b>User description of the I/O:</b><br>
 * ./.
 * <p>
 * <b>Range of the I/O value:</b><br>
 * [false; true]
 *
 * @return current value of the digital input 'InputX3Pin4'
 */
public boolean getInputX3Pin4()
{
        return getBooleanIOValue("InputX3Pin4", false);
}

/**
 * Gets the value of the <b>digital input '<i>InputX3Pin10</i>'</b>.<br>
 * <i>This method is automatically generated. Please, do not modify!</i>
 * <p>
 * <b>I/O direction and type:</b><br>
 * digital input
 * <p>
 * <b>User description of the I/O:</b><br>
 * ./.
 * <p>
 * <b>Range of the I/O value:</b><br>
 * [false; true]
 *
 * @return current value of the digital input 'InputX3Pin10'
 */
public boolean getInputX3Pin10()
{
        return getBooleanIOValue("InputX3Pin10", false);
}

/**
 * Gets the value of the <b>digital input '<i>InputX3Pin13</i>'</b>.<br>
 * <i>This method is automatically generated. Please, do not modify!</i>
```

```
     * <p>
     * <b>I/O direction and type:</b><br>
     * digital input
     * <p>
     * <b>User description of the I/O:</b><br>
     * ./.
     * <p>
     * <b>Range of the I/O value:</b><br>
     * [false; true]
     *
     * @return current value of the digital input 'InputX3Pin13'
     */
    public boolean getInputX3Pin13()
    {
            return getBooleanIOValue("InputX3Pin13", false);
    }

    /**
     * Gets the value of the <b>digital input '<i>InputX3Pin16</i>'</b>.<br>
     * <i>This method is automatically generated. Please, do not modify!</i>
     * <p>
     * <b>I/O direction and type:</b><br>
     * digital input
     * <p>
     * <b>User description of the I/O:</b><br>
     * ./.
     * <p>
     * <b>Range of the I/O value:</b><br>
     * [false; true]
     *
     * @return current value of the digital input 'InputX3Pin16'
     */
    public boolean getInputX3Pin16()
    {
            return getBooleanIOValue("InputX3Pin16", false);
    }

    /**
     * Gets the value of the <b>digital input '<i>UserButton</i>'</b>.<br>
     * <i>This method is automatically generated. Please, do not modify!</i>
     * <p>
     * <b>I/O direction and type:</b><br>
     * digital input
     * <p>
     * <b>User description of the I/O:</b><br>
     * ./.
     * <p>
     * <b>Range of the I/O value:</b><br>
     * [false; true]
     *
     * @return current value of the digital input 'UserButton'
     */
    public boolean getUserButton()
    {
            return getBooleanIOValue("UserButton", false);
    }

    /**
     * Gets the value of the <b>digital output '<i>LEDBlue</i>'</b>.<br>
     * <i>This method is automatically generated. Please, do not modify!</i>
     * <p>
     * <b>I/O direction and type:</b><br>
     * digital output
     * <p>
     * <b>User description of the I/O:</b><br>
     * ./.
     * <p>
     * <b>Range of the I/O value:</b><br>
     * [false; true]
     *
```

```
 * @return current value of the digital output 'LEDBlue'
 */
public boolean getLEDBlue()
{
        return getBooleanIOValue("LEDBlue", true);
}

/**
 * Sets the value of the <b>digital output '<i>LEDBlue</i>'</b>.<br>
 * <i>This method is automatically generated. Please, do not modify!</i>
 * <p>
 * <b>I/O direction and type:</b><br>
 * digital output
 * <p>
 * <b>User description of the I/O:</b><br>
 * ./.
 * <p>
 * <b>Range of the I/O value:</b><br>
 * [false; true]
 *
 * @param value
 *            the value, which has to be written to the digital output 'LEDBlue'
 */
public void setLEDBlue(java.lang.Boolean value)
{
        setDigitalOutput("LEDBlue", value);
}

/**
 * Gets the value of the <b>digital output '<i>SwitchOffX3Voltage</i>'</b>.<br>
 * <i>This method is automatically generated. Please, do not modify!</i>
 * <p>
 * <b>I/O direction and type:</b><br>
 * digital output
 * <p>
 * <b>User description of the I/O:</b><br>
 * ./.
 * <p>
 * <b>Range of the I/O value:</b><br>
 * [false; true]
 *
 * @return current value of the digital output 'SwitchOffX3Voltage'
 */
public boolean getSwitchOffX3Voltage()
{
        return getBooleanIOValue("SwitchOffX3Voltage", true);
}

/**
 * Sets the value of the <b>digital output '<i>SwitchOffX3Voltage</i>'</b>.<br>
 * <i>This method is automatically generated. Please, do not modify!</i>
 * <p>
 * <b>I/O direction and type:</b><br>
 * digital output
 * <p>
 * <b>User description of the I/O:</b><br>
 * ./.
 * <p>
 * <b>Range of the I/O value:</b><br>
 * [false; true]
 *
 * @param value
 *            the value, which has to be written to the digital output 'SwitchOffX3Voltage'
 */
public void setSwitchOffX3Voltage(java.lang.Boolean value)
{
        setDigitalOutput("SwitchOffX3Voltage", value);
}

/**
```

```
	 * Gets the value of the <b>digital output '<i>ChangerOpen</i>'</b>.<br>
	 * <i>This method is automatically generated. Please, do not modify!</i>
	 * <p>
	 * <b>I/O direction and type:</b><br>
	 * digital output
	 * <p>
	 * <b>User description of the I/O:</b><br>
	 * ./.
	 * <p>
	 * <b>Range of the I/O value:</b><br>
	 * [false; true]
	 *
	 * @return current value of the digital output 'ChangerOpen'
	 */
	public boolean getChangerOpen()
	{
		return getBooleanIOValue("ChangerOpen", true);
	}

	/**
	 * Sets the value of the <b>digital output '<i>ChangerOpen</i>'</b>.<br>
	 * <i>This method is automatically generated. Please, do not modify!</i>
	 * <p>
	 * <b>I/O direction and type:</b><br>
	 * digital output
	 * <p>
	 * <b>User description of the I/O:</b><br>
	 * ./.
	 * <p>
	 * <b>Range of the I/O value:</b><br>
	 * [false; true]
	 *
	 * @param value
	 *            the value, which has to be written to the digital output 'ChangerOpen'
	 */
	public void setChangerOpen(java.lang.Boolean value)
	{
		setDigitalOutput("ChangerOpen", value);
	}

	/**
	 * Gets the value of the <b>digital output '<i>ChangerClose</i>'</b>.<br>
	 * <i>This method is automatically generated. Please, do not modify!</i>
	 * <p>
	 * <b>I/O direction and type:</b><br>
	 * digital output
	 * <p>
	 * <b>User description of the I/O:</b><br>
	 * ./.
	 * <p>
	 * <b>Range of the I/O value:</b><br>
	 * [false; true]
	 *
	 * @return current value of the digital output 'ChangerClose'
	 */
	public boolean getChangerClose()
	{
		return getBooleanIOValue("ChangerClose", true);
	}

	/**
	 * Sets the value of the <b>digital output '<i>ChangerClose</i>'</b>.<br>
	 * <i>This method is automatically generated. Please, do not modify!</i>
	 * <p>
	 * <b>I/O direction and type:</b><br>
	 * digital output
	 * <p>
	 * <b>User description of the I/O:</b><br>
	 * ./.
	 * <p>
```

```
     * <b>Range of the I/O value:</b><br>
     * [false; true]
     *
     * @param value
     *            the value, which has to be written to the digital output 'ChangerClose'
     */
    public void setChangerClose(java.lang.Boolean value)
    {
            setDigitalOutput("ChangerClose", value);
    }

    /**
     * Gets the value of the <b>digital output '<i>LEDGreen</i>'</b>.<br>
     * <i>This method is automatically generated. Please, do not modify!</i>
     * <p>
     * <b>I/O direction and type:</b><br>
     * digital output
     * <p>
     * <b>User description of the I/O:</b><br>
     * ./.
     * <p>
     * <b>Range of the I/O value:</b><br>
     * [false; true]
     *
     * @return current value of the digital output 'LEDGreen'
     */
    public boolean getLEDGreen()
    {
            return getBooleanIOValue("LEDGreen", true);
    }

    /**
     * Sets the value of the <b>digital output '<i>LEDGreen</i>'</b>.<br>
     * <i>This method is automatically generated. Please, do not modify!</i>
     * <p>
     * <b>I/O direction and type:</b><br>
     * digital output
     * <p>
     * <b>User description of the I/O:</b><br>
     * ./.
     * <p>
     * <b>Range of the I/O value:</b><br>
     * [false; true]
     *
     * @param value
     *            the value, which has to be written to the digital output 'LEDGreen'
     */
    public void setLEDGreen(java.lang.Boolean value)
    {
            setDigitalOutput("LEDGreen", value);
    }

    /**
     * Gets the value of the <b>digital output '<i>LEDRed</i>'</b>.<br>
     * <i>This method is automatically generated. Please, do not modify!</i>
     * <p>
     * <b>I/O direction and type:</b><br>
     * digital output
     * <p>
     * <b>User description of the I/O:</b><br>
     * ./.
     * <p>
     * <b>Range of the I/O value:</b><br>
     * [false; true]
     *
     * @return current value of the digital output 'LEDRed'
     */
    public boolean getLEDRed()
    {
            return getBooleanIOValue("LEDRed", true);
```

```
}

/**
 * Sets the value of the <b>digital output '<i>LEDRed</i>'</b>.<br>
 * <i>This method is automatically generated. Please, do not modify!</i>
 * <p>
 * <b>I/O direction and type:</b><br>
 * digital output
 * <p>
 * <b>User description of the I/O:</b><br>
 * ./.
 * <p>
 * <b>Range of the I/O value:</b><br>
 * [false; true]
 *
 * @param value
 *            the value, which has to be written to the digital output 'LEDRed'
 */
public void setLEDRed(java.lang.Boolean value)
{
        setDigitalOutput("LEDRed", value);
}

/**
 * Gets the value of the <b>digital output '<i>GripperClose</i>'</b>.<br>
 * <i>This method is automatically generated. Please, do not modify!</i>
 * <p>
 * <b>I/O direction and type:</b><br>
 * digital output
 * <p>
 * <b>User description of the I/O:</b><br>
 * OutputX3Pin12
 * <p>
 * <b>Range of the I/O value:</b><br>
 * [false; true]
 *
 * @return current value of the digital output 'GripperClose'
 */
public boolean getGripperClose()
{
        return getBooleanIOValue("GripperClose", true);
}

/**
 * Sets the value of the <b>digital output '<i>GripperClose</i>'</b>.<br>
 * <i>This method is automatically generated. Please, do not modify!</i>
 * <p>
 * <b>I/O direction and type:</b><br>
 * digital output
 * <p>
 * <b>User description of the I/O:</b><br>
 * OutputX3Pin12
 * <p>
 * <b>Range of the I/O value:</b><br>
 * [false; true]
 *
 * @param value
 *            the value, which has to be written to the digital output 'GripperClose'
 */
public void setGripperClose(java.lang.Boolean value)
{
        setDigitalOutput("GripperClose", value);
}

/**
 * Gets the value of the <b>digital output '<i>GripperOpen</i>'</b>.<br>
 * <i>This method is automatically generated. Please, do not modify!</i>
 * <p>
 * <b>I/O direction and type:</b><br>
 * digital output
```

```
         * <p>
         * <b>User description of the I/O:</b><br>
         * OutputX3Pin2
         * <p>
         * <b>Range of the I/O value:</b><br>
         * [false; true]
         *
         * @return current value of the digital output 'GripperOpen'
         */
        public boolean getGripperOpen()
        {
                return getBooleanIOValue("GripperOpen", true);
        }

        /**
         * Sets the value of the <b>digital output '<i>GripperOpen</i>'</b>.<br>
         * <i>This method is automatically generated. Please, do not modify!</i>
         * <p>
         * <b>I/O direction and type:</b><br>
         * digital output
         * <p>
         * <b>User description of the I/O:</b><br>
         * OutputX3Pin2
         * <p>
         * <b>Range of the I/O value:</b><br>
         * [false; true]
         *
         * @param value
         *            the value, which has to be written to the digital output 'GripperOpen'
         */
        public void setGripperOpen(java.lang.Boolean value)
        {
                setDigitalOutput("GripperOpen", value);
        }

}
```

## F.2 Robot drone assembly


**Figure F.1:** Drone frame assembly at SII-Lab

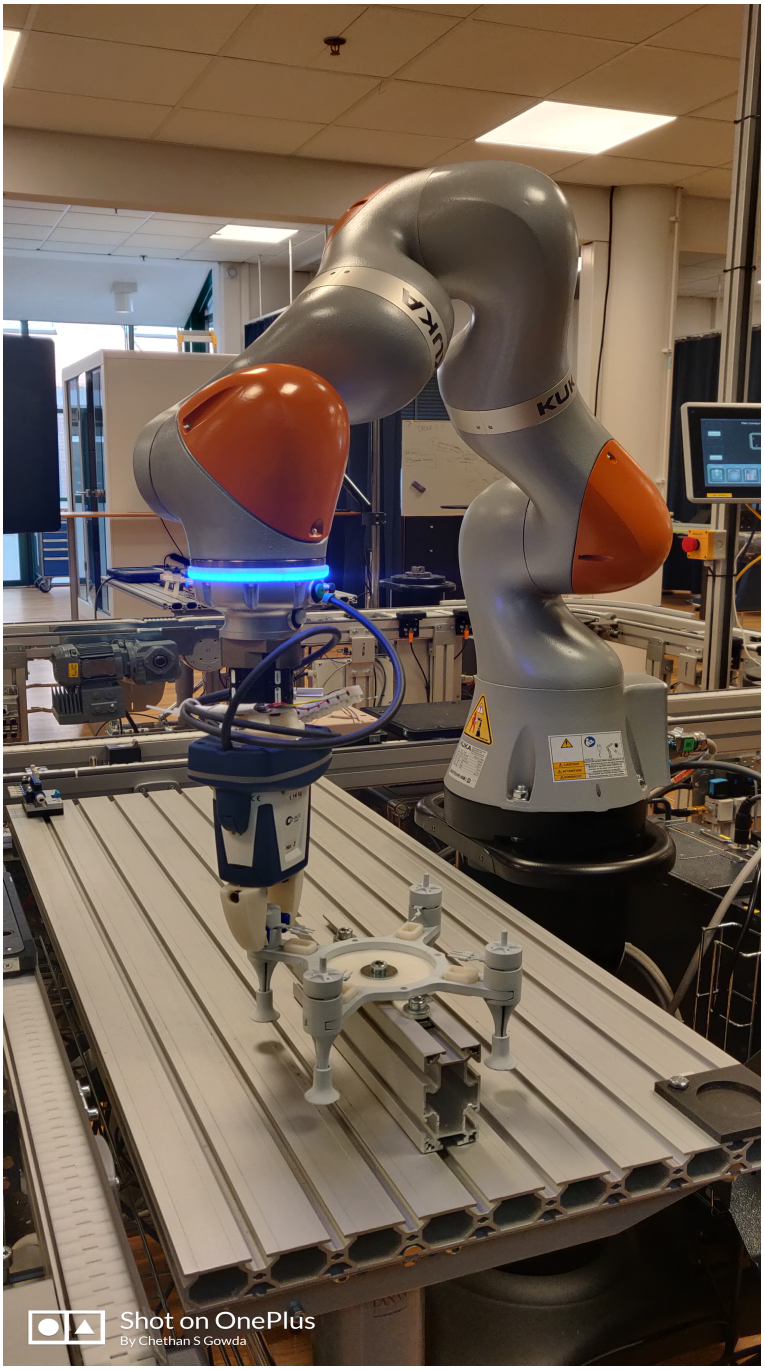



**Figure F.2:** Motor 4 assembly at SII-Lab

# G
# Appendix

## G.1 Force Torque analysis

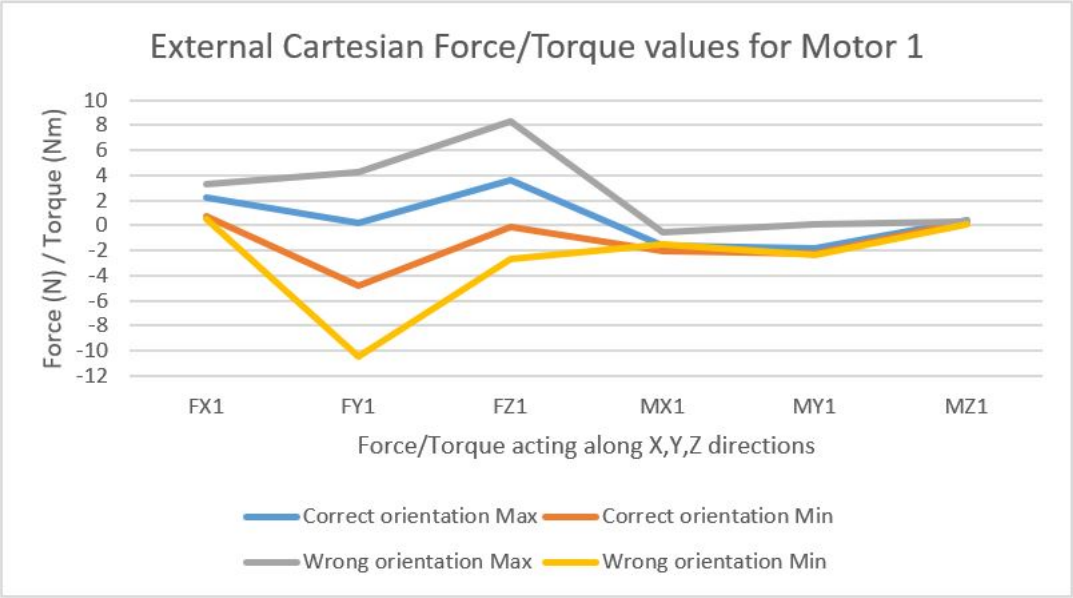
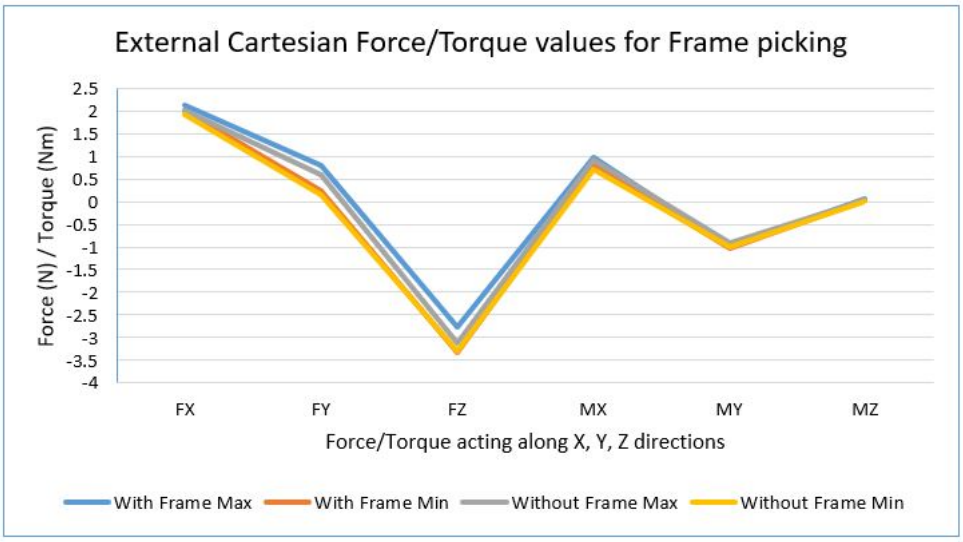
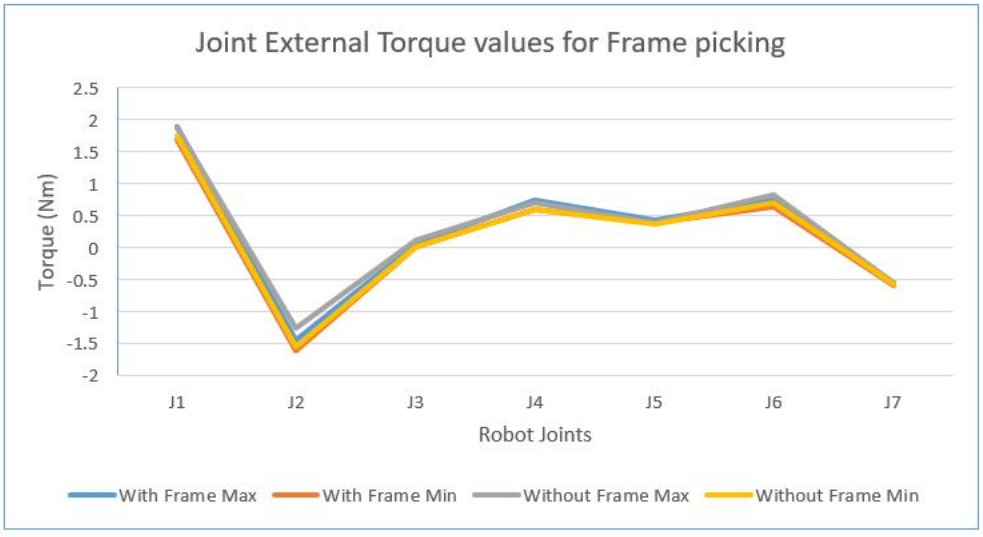



**Figure G.1:** External cartesian forces/torques for Motor 1

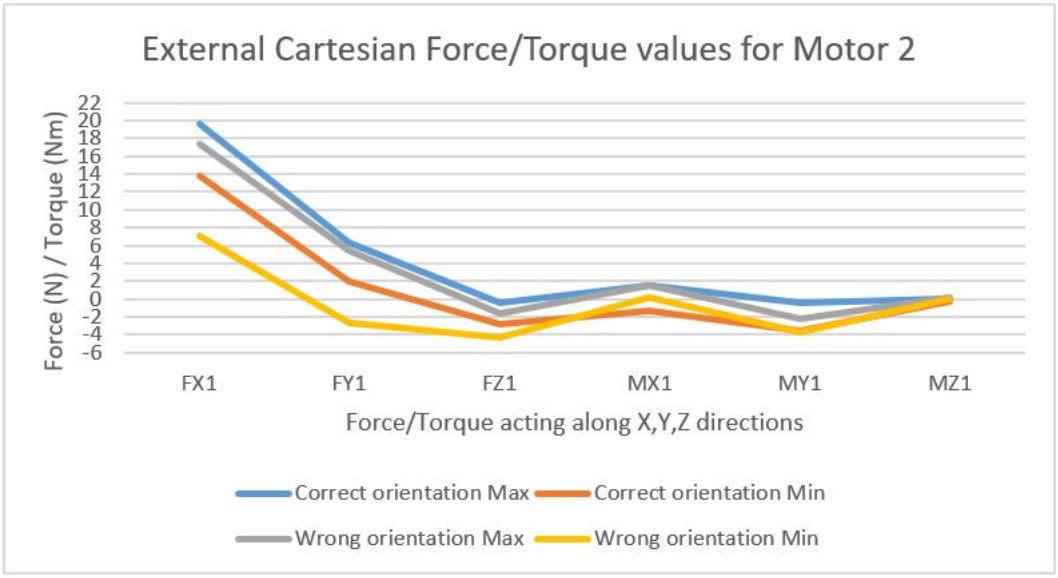



**(a)** External cartesian forces/torques for frame picking

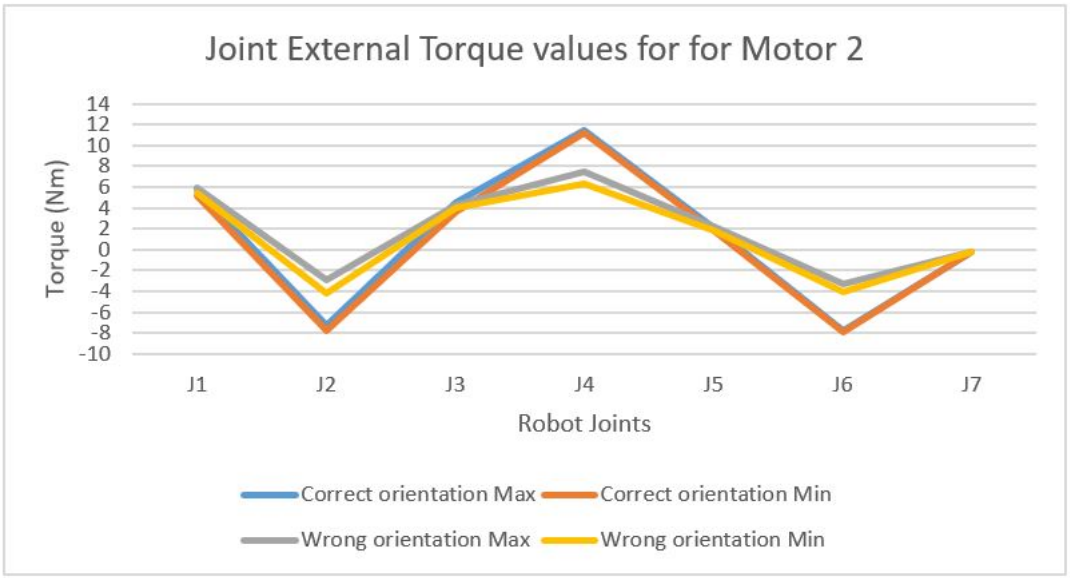



**(b)** External joint torques for frame picking

**Figure G.2:** Force and torque during drone frame picking

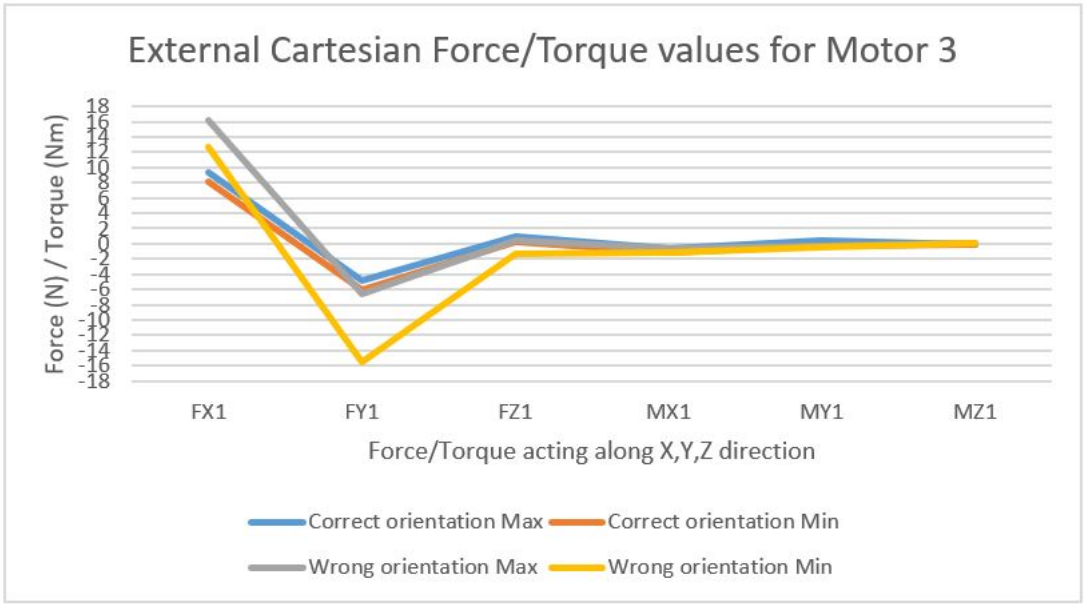



**(a)** External cartesian forces/torques for Motor 2

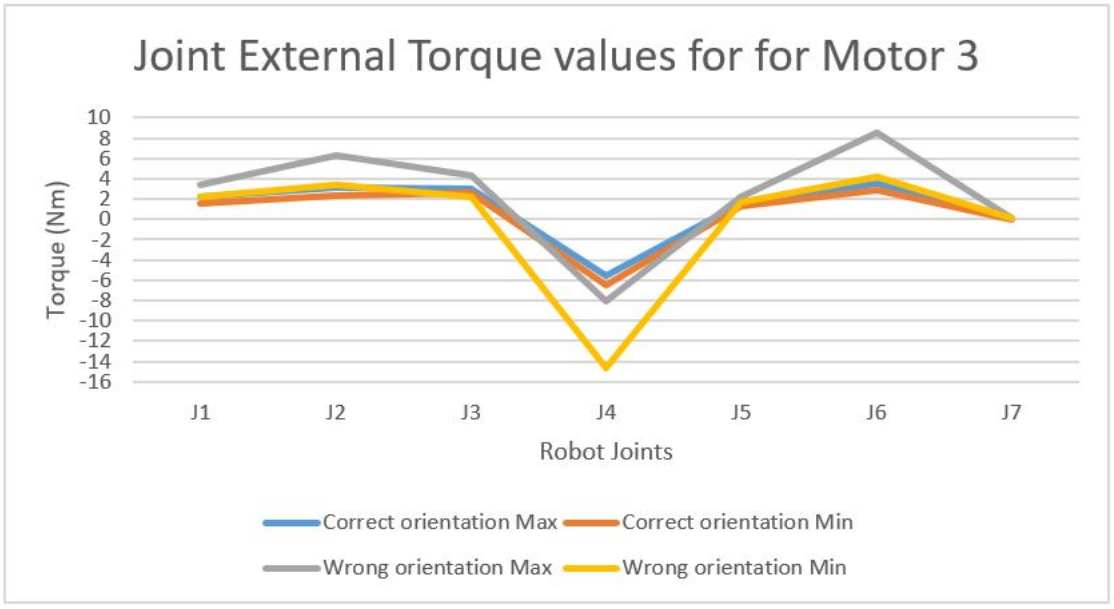



**(b)** External joint torques for Motor 2

**Figure G.3:** Force and torque during motor 2 assembly




**(a)** External cartesian forces/torques for Motor 3




**(b)** External joint torques for Motor 3

**Figure G.4:** Force and torque during motor 3 assembly

## G.2 Source code for Force torque analysis

```
package application;


import java.util.concurrent.TimeUnit;

import javax.inject.Inject;
import javax.inject.Named;

import com.kuka.generated.ioAccess.MediaFlangeIOGroup;
import com.kuka.roboticsAPI.applicationModel.RoboticsAPIApplication;
import static com.kuka.roboticsAPI.motionModel.BasicMotions.*;

import com.kuka.roboticsAPI.conditionModel.BooleanIOCondition;
import com.kuka.roboticsAPI.conditionModel.ForceComponentCondition;
import com.kuka.roboticsAPI.conditionModel.ForceCondition;
import com.kuka.roboticsAPI.conditionModel.FrameDistanceCondition;
import com.kuka.roboticsAPI.conditionModel.ICallbackAction;
import com.kuka.roboticsAPI.conditionModel.ICondition;
import com.kuka.roboticsAPI.conditionModel.IORangeCondition;
import com.kuka.roboticsAPI.conditionModel.JointTorqueCondition;
import com.kuka.roboticsAPI.conditionModel.TorqueComponentCondition;
import com.kuka.roboticsAPI.deviceModel.JointEnum;
import com.kuka.roboticsAPI.deviceModel.LBR;
import com.kuka.roboticsAPI.executionModel.IFiredConditionInfo;
import com.kuka.roboticsAPI.executionModel.IFiredTriggerInfo;
import com.kuka.roboticsAPI.geometricModel.CartDOF;
import com.kuka.roboticsAPI.geometricModel.Frame;
import com.kuka.roboticsAPI.geometricModel.Tool;
import com.kuka.roboticsAPI.geometricModel.Workpiece;
import com.kuka.roboticsAPI.geometricModel.math.CoordinateAxis;
import com.kuka.roboticsAPI.geometricModel.math.Vector;
import com.kuka.roboticsAPI.ioModel.AbstractIO;
import com.kuka.roboticsAPI.motionModel.IMotionContainer;
import com.kuka.roboticsAPI.motionModel.controlModeModel.CartesianImpedanceControlMode;
import com.kuka.roboticsAPI.motionModel.controlModeModel.CartesianSineImpedanceControlMode;
import com.kuka.roboticsAPI.sensorModel.DataRecorder;
import com.kuka.roboticsAPI.sensorModel.ForceSensorData;
import com.kuka.roboticsAPI.sensorModel.TorqueEvaluator;
import com.kuka.roboticsAPI.sensorModel.TorqueSensorData;
import com.kuka.roboticsAPI.sensorModel.TorqueStatistic;
import com.kuka.task.ITaskLogger;



/*
 * Implementation of a robot application.
 * <p>
 * The application provides a {@link RoboticsAPITask#initialize()} and a
 * {@link RoboticsAPITask#run()} method, which will be called successively in
 * the application lifecycle. The application will terminate automatically after
 * the {@link RoboticsAPITask#run()} method has finished or after stopping the
 * task. The {@link RoboticsAPITask#dispose()} method will be called, even if an
 * exception is thrown during initialization or run.
 * <p>
 * <b>It is imperative to call <code>super.dispose()</code> when overriding the
 * {@link RoboticsAPITask#dispose()} method.</b>
 * @param <SensorIOGroup>
 *
 * @see UseRoboticsAPIContext
 * @see #initialize()
 * @see #run()
 * @see #dispose()
 */


public class KukaIIWADrine_NewFixture extends RoboticsAPIApplication {
    @Inject
```

```
    private LBR lBR_iiwa_14_R820_1;
    private MediaFlangeIOGroup MFIO = null;

    @Inject
    @Named("Gripper")
    private Tool Gripper;

    @Inject
    @Named("PointerTool")
    private Tool PointerTool;

    @Inject
    @Named("DroneFrame")
    private Workpiece DroneFrame;

    @Inject
    private ITaskLogger logger;

    @Override
    public void initialize() {
    // initialize your application here
    Gripper.attachTo(lBR_iiwa_14_R820_1.getFlange());
    //PointerTool.attachTo(lBR_iiwa_14_R820_1.getFlange());

    if(MFIO == null) {
            MFIO = new MediaFlangeIOGroup(lBR_iiwa_14_R820_1.getController());
    }
}

@Override
public void run() {
    // your application execution starts here
        lightsOff();

        DataRecorder rec1 = new DataRecorder("Recording_16June.log", 150, TimeUnit.SECONDS, 50);
        rec1.addExternalJointTorque(lBR_iiwa_14_R820_1);
        rec1.addCartesianForce(Gripper.getFrame("/GripperTCP"), null);
        rec1.enable();

        lBR_iiwa_14_R820_1.move(ptpHome());
        MFIO.setLEDRed(true);

        //rec.startRecording();
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via40")));
        MFIO.setLEDRed(false);

        //PointerTool.getFrame("/PointerToolTCP").move(ptp(getApplicationData().getFrame("/Base/fixture")));
        droneFrameAssembly();
        motorOneAssembly();
        motorTwoAssembly();
        motorThreeAssembly();
        motorFourAssembly();
        pkplAssembledDrone();
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via40")));
        lBR_iiwa_14_R820_1.move(ptpHome());

        //rec.stopRecording();

        if (rec1.awaitFileAvailable(5, TimeUnit.SECONDS)){
        }

        MFIO.setLEDRed(false);
        lightsOff();
}

private void droneFrameAssembly() {

        //gripper open
        gripperRelease();
```

```
Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via1")));
Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via2")).
setJointVelocityRel(0.05));

//gripper grasp frame
gripperGrasp();

//Force and Torque data is displayed
ForceSensorData data = lBR_iiwa_14_R820_1.getExternalForceTorque(Gripper.getFrame("/GripperTCP"));
TorqueSensorData measuredata = lBR_iiwa_14_R820_1.getMeasuredTorque();
logger.info("Force & Torque :" +data);
logger.info("Measured Torques :" +measuredata);

//Force and Torque conditions are initialized
ForceComponentCondition dronegraspForce_X = new
ForceComponentCondition(Gripper.getFrame("/GripperTCP"),CoordinateAxis.X, 2.02, 3);
/*The value is inverted to set range within 2.02N - 3N
This range is obtained from testing the scenario and recording values repeatedly*/
ICondition dronegraspForceX = dronegraspForce_X.invert();
ForceComponentCondition dronegraspForce_Z = new
ForceComponentCondition(Gripper.getFrame("/GripperTCP"),CoordinateAxis.Z,-3.7, -2.5);
ICondition dronegraspForceZ = dronegraspForce_Z.invert();
TorqueComponentCondition dronegrasptorqueForce_Z = new
TorqueComponentCondition(Gripper.getFrame("/GripperTCP"),CoordinateAxis.Z, -0.15, 0.07);
ICondition dronegrasptorqueForceZ = dronegrasptorqueForce_Z.invert();
//ICondtion object for linking 2 or more conditions
ICondition DroneFrameCombi1;
DroneFrameCombi1 = dronegraspForceX.or(dronegraspForceZ.and(dronegrasptorqueForceZ));

/* Motion command is temporarily stored in IMotionContrainer.
Motion termination is performed using breakWhen command.
Information of terminated motions are stored in IFiredCondtionInfo */

IMotionContainer motionCmdDrone =
Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via3")).
breakWhen(DroneFrameCombi1));
IFiredConditionInfo DronefiredCondInfo = motionCmdDrone.getFiredBreakConditionInfo();

/* The condition which caused the termination of a motion can be requested via the method getFiredCondition().
If the requested information is not equal to null, the motion has been terminated.
The system only requests the triggered break condition in this case.
Specific operations are provided in the source code */

if(DronefiredCondInfo != null){
        ICondition DronefiredCondition = DronefiredCondInfo.getFiredCondition();

        if(DronefiredCondition.equals(DroneFrameCombi1)){

                logger.info("Drone Frame AVAILABLE - Assembly CONTINUED");
                Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via4")).
                setJointVelocityRel(0.05));
                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via5")));
                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/newvia6")));
                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/newvia8")).
                setJointVelocityRel(0.05));

                //gripper release frame
                gripperRelease();

        }
}

else{
        //gripper release frame
        gripperRelease();
        logger.info("Drone Frame NOT Available - Assembly STOPPED");
        Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via4")).
        setJointVelocityRel(0.05));
        lBR_iiwa_14_R820_1.move(ptpHome());
}
```

```
}

private void motorOneAssembly() {

        MFIO.setLEDBlue(true);
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/newvia9")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via10")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via11")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via12")).
        setJointVelocityRel(0.05));
        MFIO.setLEDBlue(false);

        //gripper grasp motor1
        gripperGrasp();

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via13")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via14")).
        setJointVelocityRel(0.05));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via15")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via16")));

        //rec.startRecording();

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via17")));

        //Force and Torque conditions are initialized
        JointTorqueCondition Motor1cond_1 = new JointTorqueCondition(JointEnum.J1, -2, 15);
        ForceComponentCondition Motor1Force_X = new
        ForceComponentCondition(Gripper.getFrame("/GripperTCP"),CoordinateAxis.X, 0, 8);
        //Values are inverted to be in range of 0-8
        ICondition Motor1ForceX = Motor1Force_X.invert();
        ForceComponentCondition Motor1Force_Z = new
        ForceComponentCondition(Gripper.getFrame("/GripperTCP"),CoordinateAxis.Z, -0.12, 4.05);
        ICondition Motor1ForceZ = Motor1Force_Z.invert();
        ForceComponentCondition Motor1Force_Y = new
        ForceComponentCondition(Gripper.getFrame("/GripperTCP"),CoordinateAxis.Y, -4.85, 0.2);
        ICondition Motor1ForceY = Motor1Force_Y.invert();
        TorqueComponentCondition Motor1Torque_Y = new
        TorqueComponentCondition(Gripper.getFrame("/GripperTCP"),CoordinateAxis.Y, -2.25, -1.85);
        ICondition Motor1TorqueY = Motor1Torque_Y.invert();
        //ICondition object for linking 2 or more conditions
        ICondition Motor1Combi1;
        ICondition Motor1Combi2;
        Motor1Combi2 = (Motor1ForceZ.or(Motor1ForceX)).and(Motor1ForceY.or(Motor1TorqueY));
        Motor1Combi1 = Motor1ForceX.and(Motor1ForceY);

        /* Motion command is stored in IMontionContainer.
        breakWhen command is used to terminate the motion.
        Motions are terminated when ICondition is met */

        IMotionContainer motionCmdMotor1 =
        Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/newvia18_4")).
        breakWhen(Motor1Combi1));
        IFiredConditionInfo motor1firedCondInfo = motionCmdMotor1.getFiredBreakConditionInfo();

        //ForceData
        ForceSensorData Motor1data = lBR_iiwa_14_R820_1.getExternalForceTorque(Gripper.getFrame("/GripperTCP"));
        TorqueSensorData Motor1measuredata = lBR_iiwa_14_R820_1.getMeasuredTorque();
        logger.info("Force & Torque :" +Motor1data);
        logger.info("Measured Torques :" +Motor1measuredata);

        //Getting at newvia18_4
        TorqueSensorData externalData = lBR_iiwa_14_R820_1.getExternalTorque();
        double[] externalTorques = externalData.getTorqueValues();
        double torqueA1 = externalData.getSingleTorqueValue(JointEnum.J1);
        logger.info("Currently measured torque for joint 1 [Nm]:" +torqueA1);
        logger.info("Currently measured torque for joints [Nm]:" +externalTorques);

        /* Specific operation are provided for temination
        of motion via getFiredCondition */
```

```
        if(motor1firedCondInfo != null){
                ICondition Motor1firedCondition = motor1firedCondInfo.getFiredCondition();

                if(Motor1firedCondition.equals(Motor1Combi1)){

                        logger.info ("CORRECT orientation");
                        Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/newvia18_4")));
                        //New Point
                        Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via18")));
                                logger.info("Assembly COMPLETE");
                        //gripper release motor1
                        gripperRelease();
                        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via19")));
                        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via20")));

                }
        }

        else{
                logger.info ("WRONG orientation");
                Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via18")));
                gripperRelease();
                Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/newvia18_1")));
                Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/newvia18")).
                setJointVelocityRel(0.05));


                //gripper grasp motor1
                gripperGrasp();

                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/newvia18_2")).
                setJointVelocityRel(0.05));

                //gripper release motor1
                gripperRelease();

                Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/newvia18_3")));
                        logger.info("Assembly CORRECTED");
        }
}

private void motorTwoAssembly() {

        gripperRelease();

        CartesianImpedanceControlMode cartImpCtrlMode = new CartesianImpedanceControlMode();
        cartImpCtrlMode.parametrize(CartDOF.X, CartDOF.Y).setStiffness(5000.0);
        //cartImpCtrlMode.parametrize(CartDOF.Z).setStiffness(1.0);
        cartImpCtrlMode.parametrize(CartDOF.ROT).setStiffness(300.0);
        cartImpCtrlMode.parametrize(CartDOF.ALL).setDamping(0.7);
        cartImpCtrlMode.setMaxPathDeviation(8.0, 8.0, 8.0, 2.0, 2.0, 2.0);
        cartImpCtrlMode.parametrize(CartDOF.X, CartDOF.Y).setAdditionalControlForce(-35.0);

        CartesianSineImpedanceControlMode sineMode = new CartesianSineImpedanceControlMode();
        sineMode = CartesianSineImpedanceControlMode.createDesiredForce(CartDOF.X, -20, 3000);


        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via19")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via20")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via44")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via21")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via22")).
        setJointVelocityRel(0.05));

        //gripper grasp motor2
        gripperGrasp();

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via23")).
        setJointVelocityRel(0.05));
```

```
Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via24")));
Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via74")));
Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via25")));
Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via26")));

ForceSensorData Motor2data1 = lBR_iiwa_14_R820_1.getExternalForceTorque(Gripper.getFrame("/GripperTCP"));
TorqueSensorData Motor2measuredata1 = lBR_iiwa_14_R820_1.getExternalTorque();
logger.info("Force & Torque :" +Motor2data1);
logger.info("External Torques :" +Motor2measuredata1);

JointTorqueCondition Motor2cond_1 = new JointTorqueCondition(JointEnum.J2, -8, -4);
ICondition Motor2Joint2 = Motor2cond_1.invert();
JointTorqueCondition Motor2cond_2 = new JointTorqueCondition(JointEnum.J4, 10, 13);
ICondition Motor2Joint4 = Motor2cond_2.invert();
JointTorqueCondition Motor2cond_3 = new JointTorqueCondition(JointEnum.J6, -10, -6);
ICondition Motor2Joint6 = Motor2cond_3.invert();
ForceComponentCondition Motor2Force_X = new
ForceComponentCondition(Gripper.getFrame("/GripperTCP"),CoordinateAxis.X, 9, 14);
ICondition Motor2ForceX = Motor2Force_X.invert();
ForceComponentCondition Motor2Force_Z = new
ForceComponentCondition(Gripper.getFrame("/GripperTCP"),CoordinateAxis.Z, -10, -1);
ICondition Motor2ForceZ = Motor2Force_Z.invert();
ForceComponentCondition Motor2Force_Y = new
ForceComponentCondition(Gripper.getFrame("/GripperTCP"),CoordinateAxis.Y, 8, 12);
ICondition Motor2ForceY = Motor2Force_Y.invert();
TorqueComponentCondition Motor2Torque_Y = new
TorqueComponentCondition(Gripper.getFrame("/GripperTCP),CoordinateAxis.Y, -2.25, -1.85);
ICondition Motor2TorqueY = Motor2Torque_Y.invert();

ICondition Motor2Combi1;
ICondition Motor2Combi2;
//Motor2Combi1 = (Motor1ForceZ.or(Motor1ForceX)).and(Motor1ForceY.or(Motor1TorqueY));
Motor2Combi1 = Motor2ForceX.and(Motor2ForceY);
Motor2Combi2 =Motor2ForceX.and((Motor2ForceY).and(Motor2ForceZ));
IMotionContainer motionCmdMotor2 =
Gripper.getFrame("/GripperTCP").move(lin(getApplicatioData().getFrame("/Base/via26_1")).
setMode(cartImpCtrlMode).breakWhen(Motor2Combi2));
IFiredConditionInfo motor2firedCondInfo = motionCmdMotor2.getFiredBreakConditionInfo();

//Force & Torque Data
ForceSensorData Motor2data = lBR_iiwa_14_R820_1.getExternalForceTorque(Gripper.getFrame("/GripperTCP"));
TorqueSensorData Motor2measuredata = lBR_iiwa_14_R820_1.getExternalTorque();
logger.info("Force & Torque :" +Motor2data);
logger.info("External Torques :" +Motor2measuredata);

if(motor2firedCondInfo != null){
        ICondition Motor2firedCondition = motor2firedCondInfo.getFiredCondition();

    if(Motor2firedCondition.equals(Motor2Combi2)){

                logger.info ("CORRECT orientation");
                Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via26_1")));
                Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via27")));
                logger.info("Assembly COMPLETE");

                //gripper release motor1
                gripperRelease();


        }
}

else{

        logger.info ("WRONG orientation");
        Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via27")));

        //gripper release motor1
        gripperRelease();
        Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via27_1")));
```

```
                Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via27_2")).
                setJointVelocityRel(0.05));

                //gripper grasp motor1
                gripperGrasp();

                Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via27_3")).
                setJointVelocityRel(0.05));

                //gripper release motor1
                gripperRelease();

                Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via27_4")));
                logger.info("Assembly CORRECTED");


        }
}

private void motorThreeAssembly() {

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via46")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via47")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via48")).
        setJointVelocityRel(0.05));

        //gripper grasp motor3
        gripperGrasp();

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via49")).
        setJointVelocityRel(0.05));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via50")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via")));
        //Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via52")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via53")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via54")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via55")));

        JointTorqueCondition Motor3cond_1 = new JointTorqueCondition(JointEnum.J2, -9, -7);
        ICondition Motor3Joint2 = Motor3cond_1.invert();
        JointTorqueCondition Motor3cond_2 = new JointTorqueCondition(JointEnum.J4, -7, -5);
        ICondition Motor3Joint4 = Motor3cond_2.invert();
        JointTorqueCondition Motor3cond_3 = new JointTorqueCondition(JointEnum.J6, 2.8, 3.5);
        ICondition Motor3Joint6 = Motor3cond_3.invert();
        ForceComponentCondition Motor3Force_X = new
        ForceComponentCondition(Gripper.getFrame("/GripperTCP"),CoordinateAxis.X, 5, 10);
        ICondition Motor3ForceX = Motor3Force_X.invert();
        ForceComponentCondition Motor3Force_Z = new
        ForceComponentCondition(Gripper.getFrame("/GripperTCP"),CoordinateAxis.Z, -0.2, 4.05);
        ICondition Motor3ForceZ = Motor3Force_Z.invert();
        ForceComponentCondition Motor3Force_Y = new
        ForceComponentCondition(Gripper.getFrame("/GripperTCP"),CoordinateAxis.Y, -4, 8);
        ICondition Motor3ForceY = Motor3Force_Y.invert();
        TorqueComponentCondition Motor3Torque_Y = new
        TorqueComponentCondition(Gripper.getFrame("/GripperTCP"),CoordinateAxis.Y, -2.25, -1.85);
        ICondition Motor3TorqueY = Motor3Torque_Y.invert();

        ICondition Motor3Combi1;
        ICondition Motor3Combi2;

        Motor3Combi1 = Motor3ForceX.and(Motor3ForceY);
        Motor3Combi2= Motor3ForceX.and(Motor3ForceZ.and(Motor3ForceY));

        IMotionContainer motionCmdMotor3 =
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via55_1")).
        breakWhen(Motor3Combi2));
        IFiredConditionInfo motor3firedCondInfo = motionCmdMotor3.getFiredBreakConditionInfo();

        //Force & Torque Data with Inaccuracies
                ForceSensorData Motor3data = lBR_iiwa_14_R820_1.getExternalForceTorque(Gripper.getFrame("/GripperTCP"));
```

```
                TorqueSensorData Motor3measuredata = lBR_iiwa_14_R820_1.getExternalTorque();
                Vector M3force = Motor3data.getForceInaccuracy();
                Vector M3torque = Motor3data.getTorqueInaccuracy();
                logger.info("Force & Torque :" +Motor3data);
                logger.info("External Torques :" +Motor3measuredata);
                logger.info("Force Inaccuracy :" +M3force);
                logger.info("Torque Inaccuracy :" +M3torque);

                if(motor3firedCondInfo != null){
                        ICondition Motor3firedCondition = motor3firedCondInfo.getFiredCondition();

                    if(Motor3firedCondition.equals(Motor3Combi2)){
                            logger.info ("CORRECT orientation");
                            Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via55_1")));
                            Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via56")));
                                logger.info("Assembly COMPLETE");

                            //gripper release motor1
                            gripperRelease();

                            Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via57")).
                            setJointVelocityRel(0.05));
                            Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via58")));
                            }
                }

                else{
                        logger.info ("WRONG orientation");
                        Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via56")));
                                gripperRelease();
                        Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via56_1")));
                        Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via56_2")).
                        setJointVelocityRel(0.05));


                        //gripper grasp motor1
                        gripperGrasp();

                        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via56_3")).
                        setJointVelocityRel(0.05));

                        //gripper release motor1
                        gripperRelease();

                        Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via56_4")));
                        logger.info("Assembly CORRECTED");
                }

}

private void motorFourAssembly() {

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via58")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via59")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via60")));

        //gripper grasp motor4
        gripperGrasp();

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via61")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via62")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via29")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via30")));

        //gripper release motor4
        gripperRelease();

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via28")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via32")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via31")));
```

```
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via33")));

        //gripper re-grasp motor4
        gripperGrasp();

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via34")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via35")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via36")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via38")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via39")));
        Gripper.getFrame("/GripperTCP").move(lin(getApplicationData().getFrame("/Base/via37")));

        //gripper re-release motor4
        gripperRelease();
}

private void pkplAssembledDrone() {
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via67")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/newvia70")));

        //gripper grasp drone
        gripperGrasp();

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/newvia71")));
        //Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via72")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/newvia73")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via75")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via76")));

        //gripper release drone
        gripperRelease();

        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via77")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via78")));
        Gripper.getFrame("/GripperTCP").move(ptp(getApplicationData().getFrame("/Base/via40")));
}

private void gripperGrasp() {
        MFIO.setLEDBlue(false);
        MFIO.setLEDGreen(true);
        MFIO.setGripperOpen(false);
        MFIO.setGripperClose(true);
}

private void gripperRelease() {
        MFIO.setGripperClose(false);
        MFIO.setGripperOpen(true);
        MFIO.setLEDGreen(false);
        MFIO.setLEDBlue(true);
}

private void lightsOff(){
        MFIO.setLEDGreen(false);
        MFIO.setLEDBlue(false);
        MFIO.setLEDRed(false);
}

}
```

# G.3 Sensitivity analysis

## G.3.1 Cartesian forces and torque values

**Table G.1:** Cartesian force and torque values recorded at coordinate points

| Trial No. | Weight (g) | Flange TCP Position | Cartesian Forces (N) | | | Cartesian Torques (Nm) | | |
|---|---|---|---|---|---|---|---|---|
| | | | Fx | Fy | Fz | Mx | My | Mz |
| 1 | 0 | X axis Downwards | 22.65 | -2.75 | 1.64 | -0.93 | 0.27 | -0.06 |
| | | Y axis Downwards | 2.62 | 20.77 | -0.22 | -0.71 | -0.94 | -0.85 |
| | | Z axis Downwards | 1.49 | -9.32 | -20.01 | -2.68 | -0.85 | 0.36 |
| 2 | | X axis Downwards | 23.27 | -3.69 | 1.93 | -1.12 | 0.16 | -0.06 |
| | | Y axis Downwards | 2.25 | 20.67 | -0.71 | -0.81 | -0.9 | -0.9 |
| | | Z axis Downwards | 1.68 | -8.71 | -20.16 | -2.68 | -0.92 | 0.4 |
| 3 | | X axis Downwards | 23.12 | -2.62 | 1.95 | -0.91 | 0.18 | -0.06 |
| | | Y axis Downwards | 3.08 | 19.86 | 0.1 | -0.86 | -1.21 | -0.98 |
| | | Z axis Downwards | 1.84 | -8.58 | -19.92 | -2.57 | -0.95 | 0.45 |
| 4 | 1760 | X axis Downwards | 5.7 | -1.48 | 2.01 | -1.03 | -1.81 | -0.52 |
| | | Y axis Downwards | 1.15 | 3.32 | -0.54 | 1.12 | -0.92 | -0.26 |
| | | Z axis Downwards | 1.54 | -8.36 | -3.57 | -2.8 | -0.21 | 0.38 |
| 5 | | X axis Downwards | 5.94 | -1.75 | 2.06 | -1.09 | -1.8 | -0.45 |
| | | Y axis Downwards | 1.5 | 3.54 | 0.29 | 1.48 | -0.94 | -0.24 |
| | | Z axis Downwards | 1.48 | -8.76 | -3.42 | -2.81 | -0.21 | 0.4 |
| 6 | | X axis Downwards | 5.75 | -1.65 | 1.94 | -1.06 | -1.82 | -0.53 |
| | | Y axis Downwards | 1.5 | 3.66 | 0.48 | 1.53 | -0.93 | -0.25 |
| | | Z axis Downwards | 1.25 | -8.12 | -3.12 | -2.8 | -0.14 | 0.37 |
| 7 | 10 | X axis Downwards | 22.86 | -2.37 | 1.78 | -0.92 | 0.24 | -0.09 |
| | | Y axis Downwards | 1.52 | 20.72 | 0.09 | -0.66 | -0.72 | -0.83 |
| | | Z axis Downwards | 1.34 | -8.98 | -20.03 | -2.6 | -0.85 | 0.43 |
| 8 | | X axis Downwards | 22.88 | -3.19 | 1.8 | -1.03 | 0.19 | -0.08 |
| | | Y axis Downwards | 2.47 | 20.41 | -0.36 | 0.74 | -0.98 | -0.89 |
| | | Z axis Downwards | 1.94 | -8.57 | -19.89 | -2.63 | -0.95 | 0.44 |
| 9 | 250 | X axis Downwards | 20.81 | -2.92 | 1.69 | -1.05 | 0.05 | -0.06 |
| | | Y axis Downwards | 1.36 | 18.49 | 0.24 | -0.41 | -0.74 | -0.82 |
| | | Z axis Downwards | 1.41 | -8.54 | -17.54 | -2.5 | -0.74 | 0.38 |
| 10 | | X axis Downwards | 20.69 | -2.36 | 2.17 | -0.91 | -0.02 | -0.09 |
| | | Y axis Downwards | 3.1 | 18.23 | -0.91 | -0.58 | -1.27 | -0.98 |
| | | Z axis Downwards | 1.43 | -8.61 | -17.64 | -2.55 | -0.78 | 0.4 |
| 11 | 360 | X axis Downwards | 19.58 | -2.22 | 1.99 | -0.97 | -0.01 | -0.07 |
| | | Y axis Downwards | 2.66 | 17.7 | -0.34 | -0.28 | -1.09 | -0.9 |
| | | Z axis Downwards | 1.22 | -9.29 | -16.73 | -2.74 | -0.66 | 0.4 |
| 12 | | X axis Downwards | 19.23 | -2.07 | 1.55 | -0.89 | 0.03 | -0.08 |
| | | Y axis Downwards | 2.2 | 17.76 | 0.1 | -0.32 | -1.02 | -0.9 |
| | | Z axis Downwards | 1.45 | -8.52 | -16.42 | -2.59 | -0.76 | 0.43 |
| 13 | 90 | X axis Downwards | 22.21 | -2.01 | 2.02 | -0.86 | 0.12 | -0.09 |
| | | Y axis Downwards | 1.46 | 20.04 | 0.2 | -0.49 | -0.77 | -0.82 |
| | | Z axis Downwards | 1.88 | -9.29 | -18.99 | -2.68 | -0.99 | 0.49 |
| 14 | | X axis Downwards | 21.63 | -2.18 | 1.45 | -0.8 | 0.17 | -0.06 |
| | | Y axis Downwards | 2.9 | 19.93 | -0.65 | -0.63 | -1.29 | -0.9 |
| | | Z axis Downwards | 1.48 | -8.85 | -18.87 | -2.59 | -0.74 | 0.42 |
| 15 | 154 | X axis Downwards | 21.02 | -2.72 | 1.45 | -0.78 | 0.1 | -0.12 |
| | | Y axis Downwards | 2.39 | 19.34 | -0.46 | -0.59 | -1.09 | -0.93 |
| | | Z axis Downwards | 1.65 | -7.94 | -18.2 | -2.5 | -0.8 | 0.47 |
| 16 | | X axis Downwards | 21.19 | -2.77 | 1.59 | -0.86 | 0.07 | -0.1 |
| | | Y axis Downwards | 2.25 | 19.52 | -0.34 | -0.47 | -1.05 | -0.92 |
| | | Z axis Downwards | 1.36 | -9.05 | -18.5 | -2.65 | -0.62 | 0.45 |
| 17 | 560 | X axis Downwards | 17.39 | -2.37 | 1.46 | -0.86 | -0.23 | -0.07 |
| | | Y axis Downwards | 1.81 | 15.71 | -0.12 | -0.15 | -0.97 | -0.83 |
| | | Z axis Downwards | 1.41 | -8.74 | -14.87 | -2.64 | -0.64 | 0.43 |
| 18 | | X axis Downwards | 17.77 | -3.18 | 1.42 | -1 | -0.28 | -0.05 |
| | | Y axis Downwards | 2.12 | 15.5 | 0.12 | -0.4 | -1.02 | -0.82 |
| | | Z axis Downwards | 1.81 | -8.47 | -14.49 | -2.62 | -0.6 | 0.45 |

### G.3.2 Plots of continuous recording

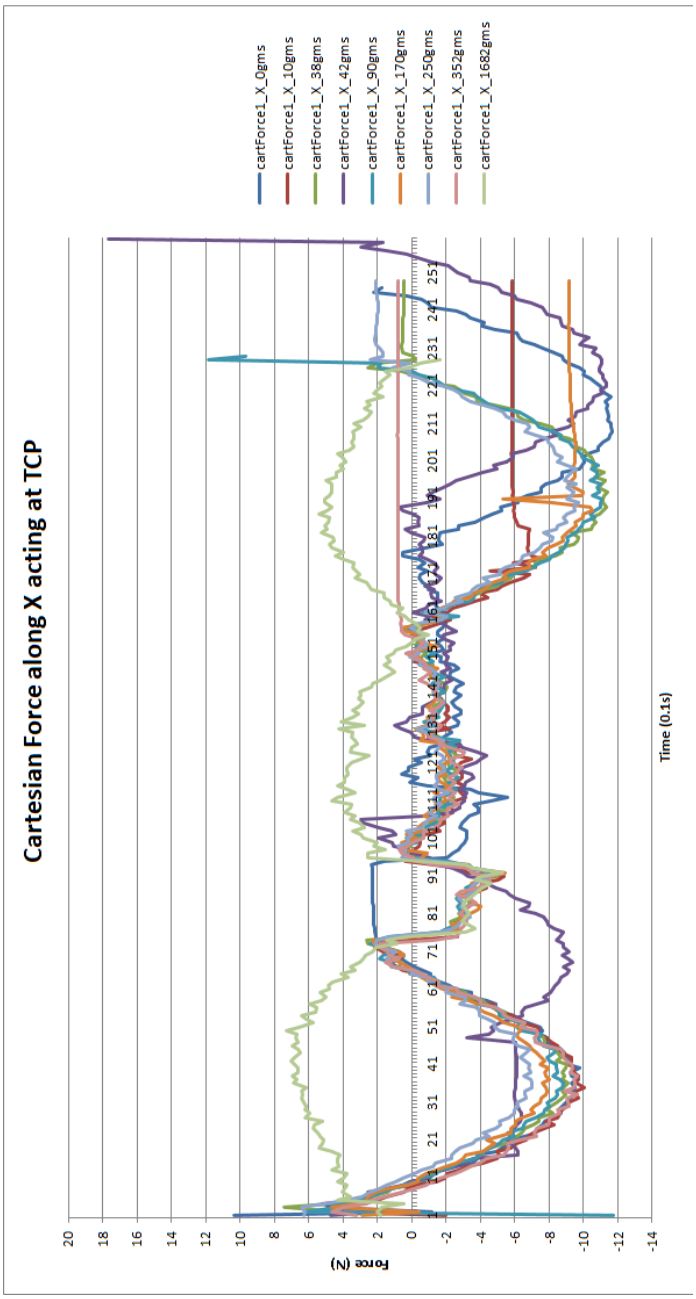



**Figure G.5:** Cartesian forces along X

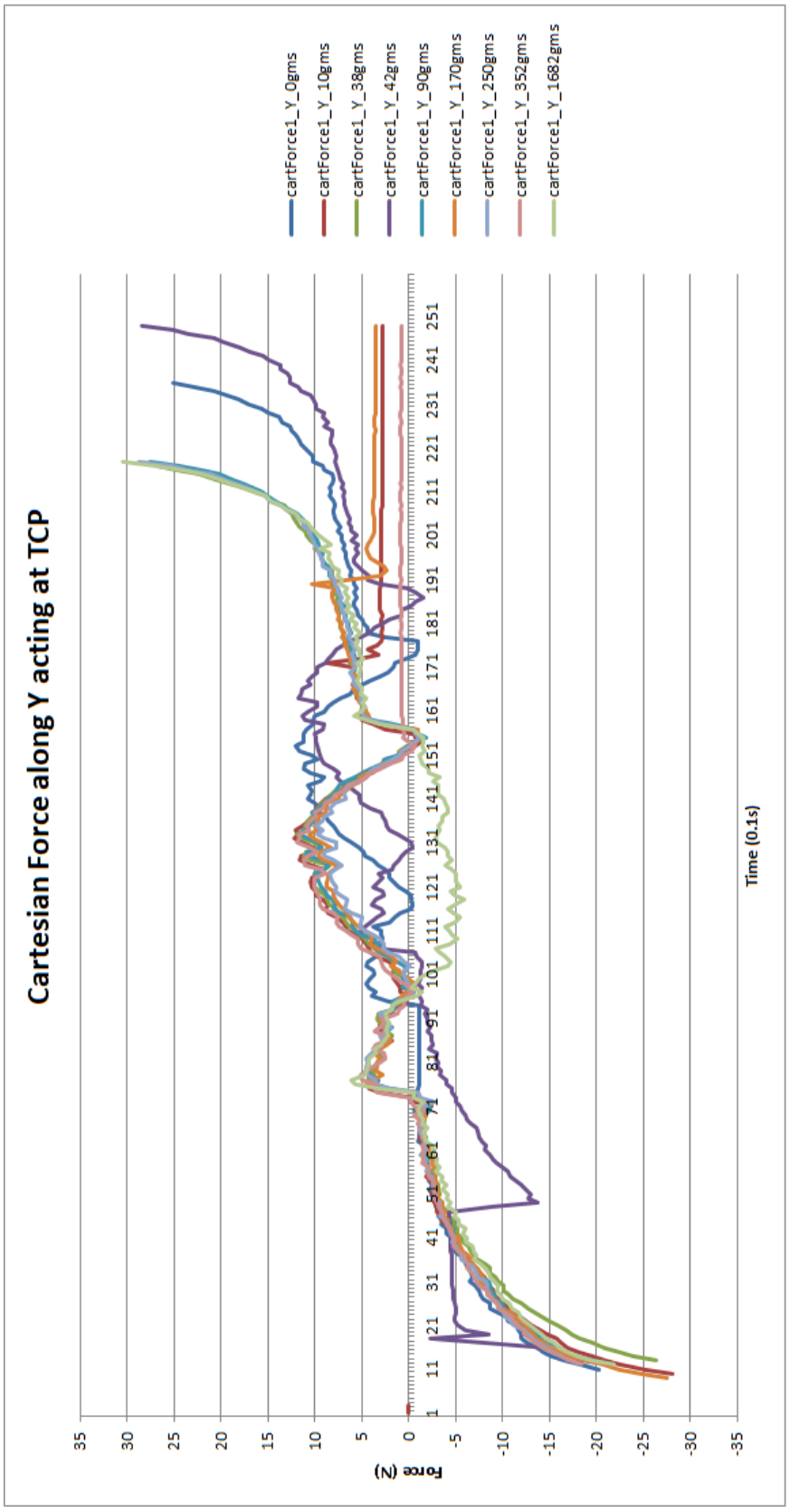



**Figure G.6:** Cartesian forces along Y

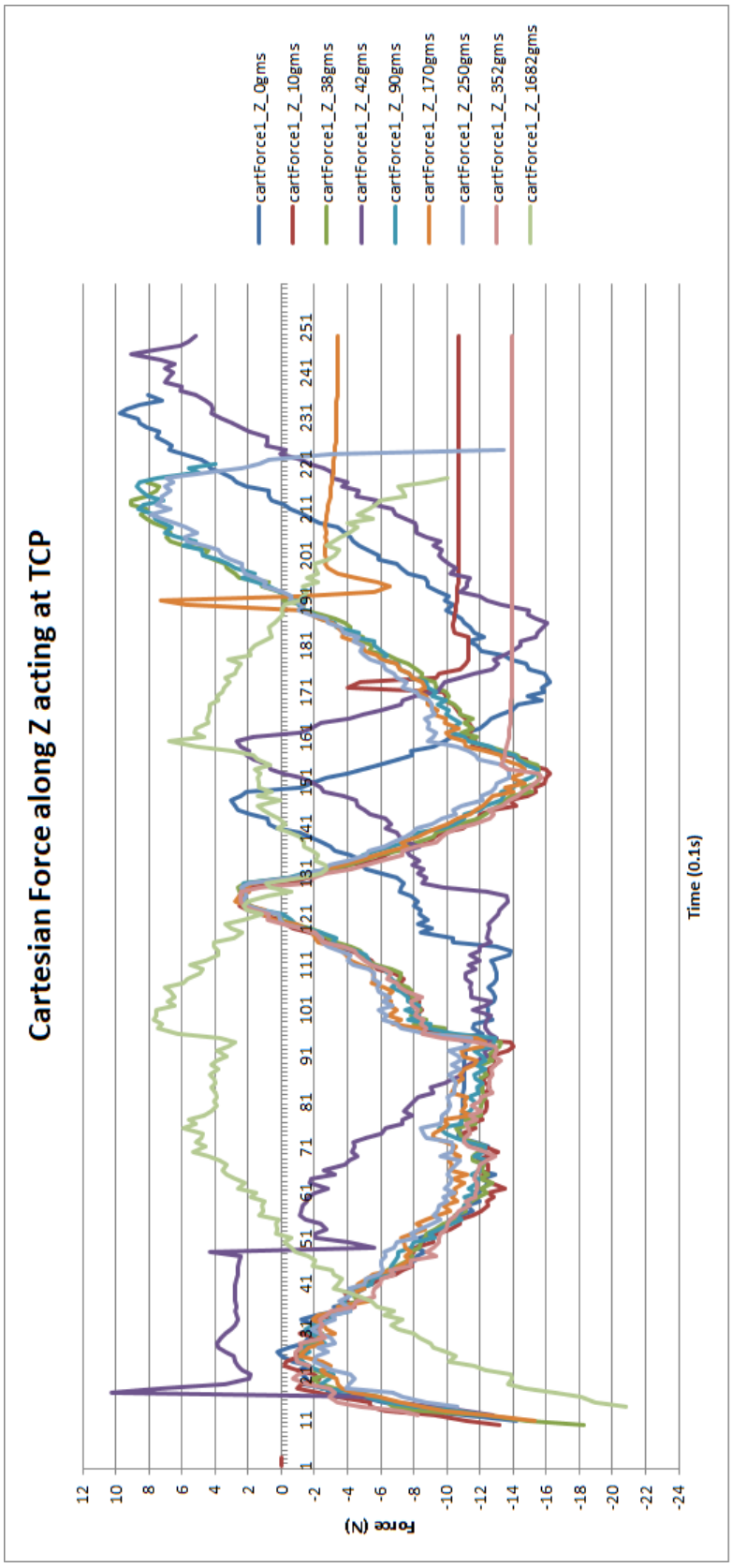



**Figure G.7:** Cartesian forces along Z

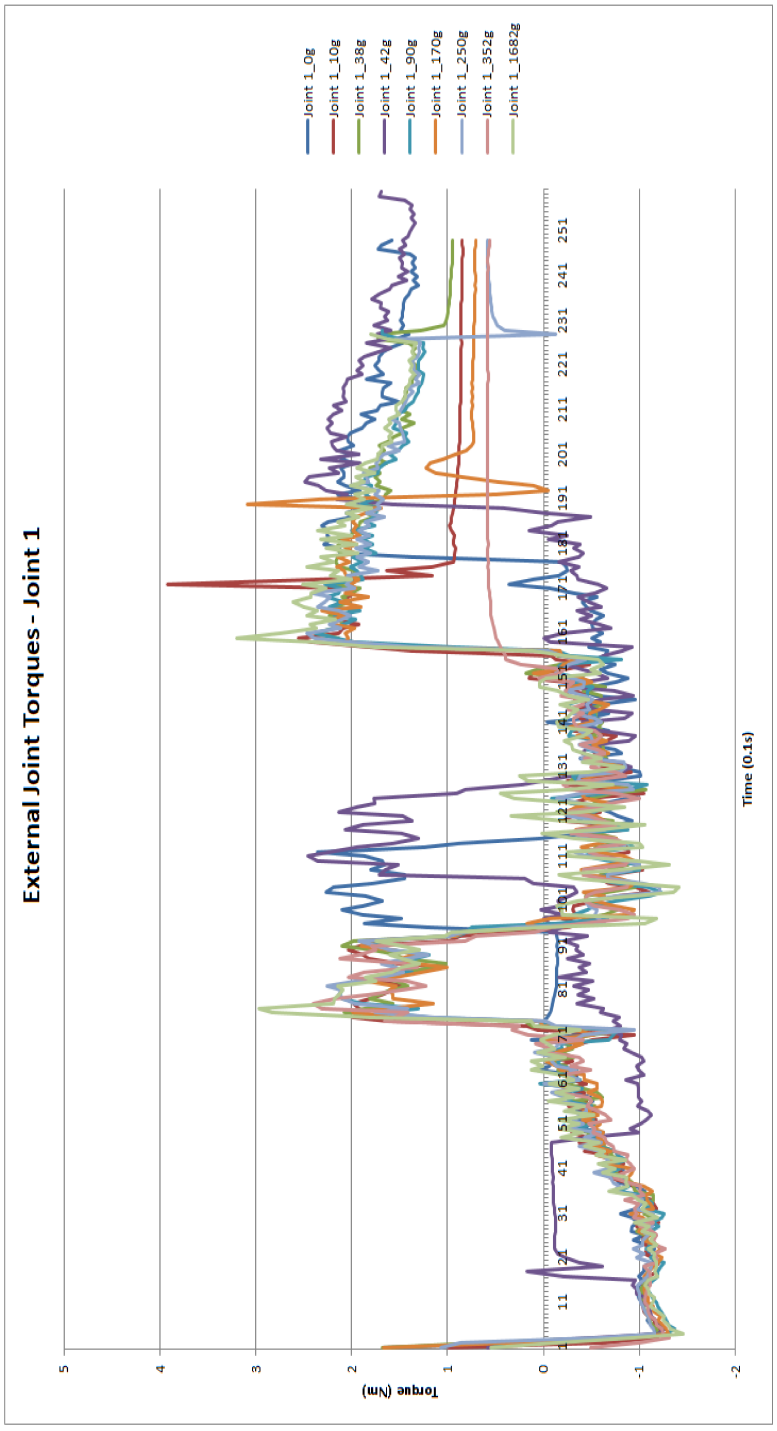



**Figure G.8:** External joint torque at Joint 1

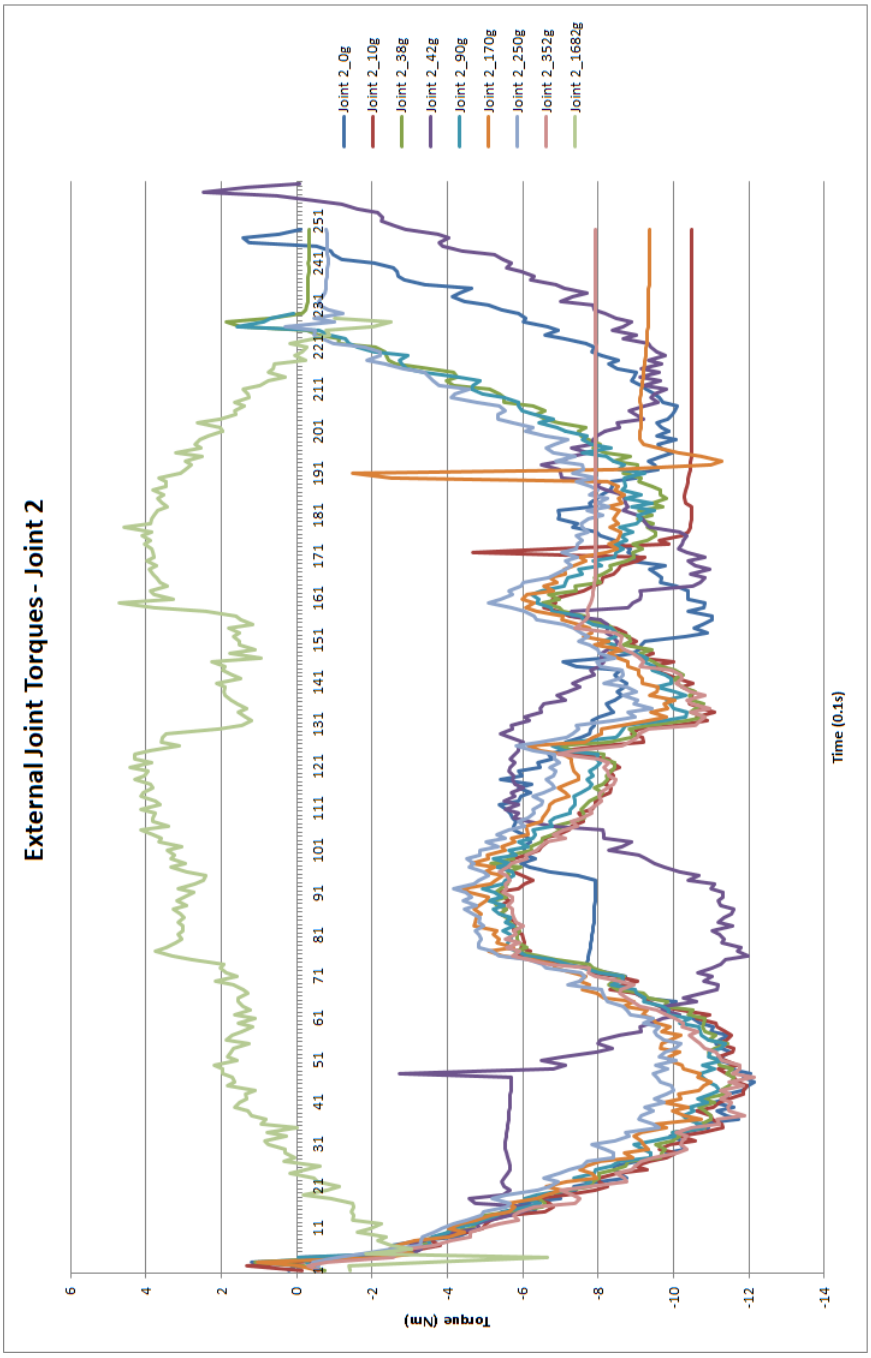



**Figure G.9:** External joint torque at Joint 2

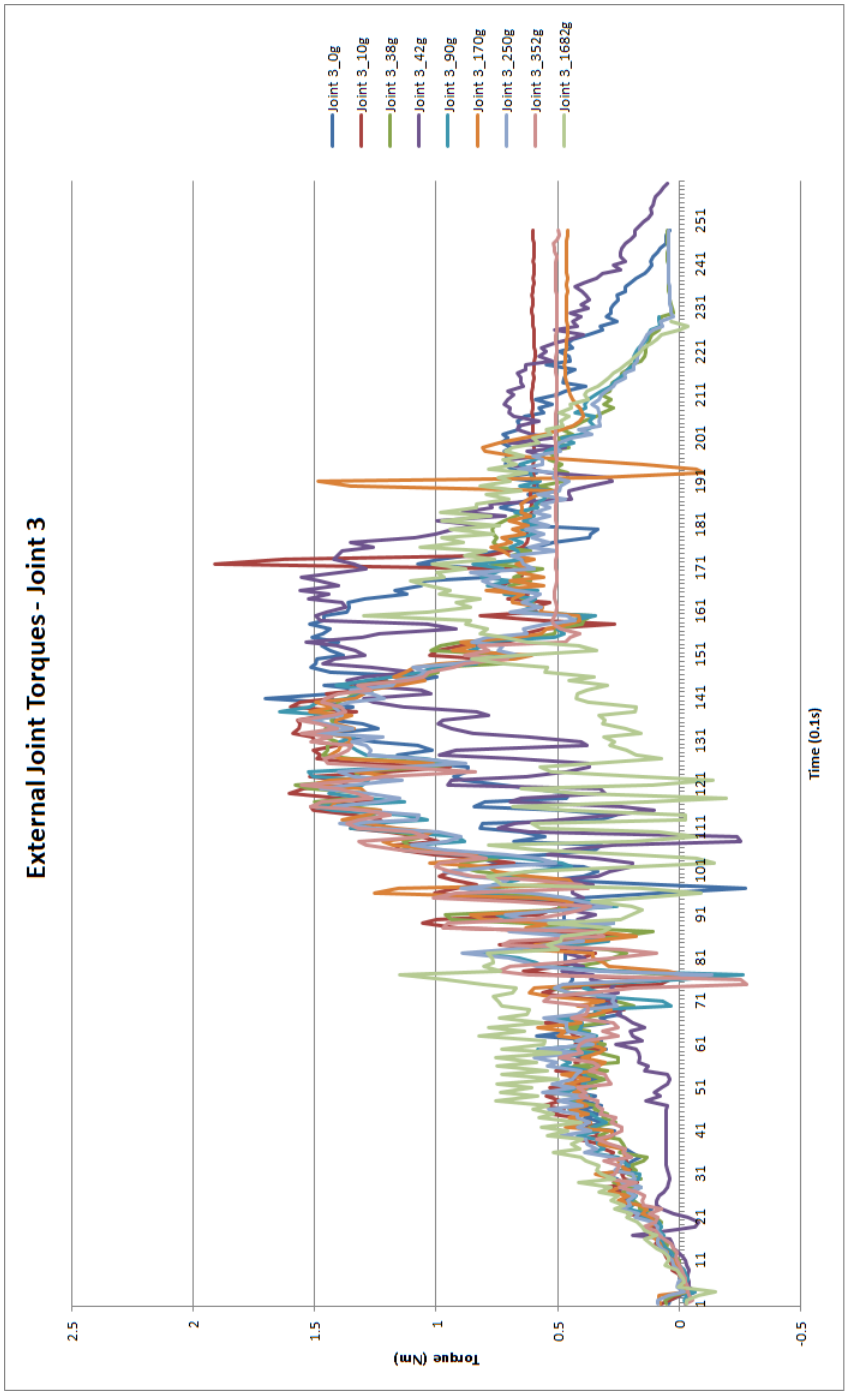



**Figure G.10:** External joint torque at Joint 3

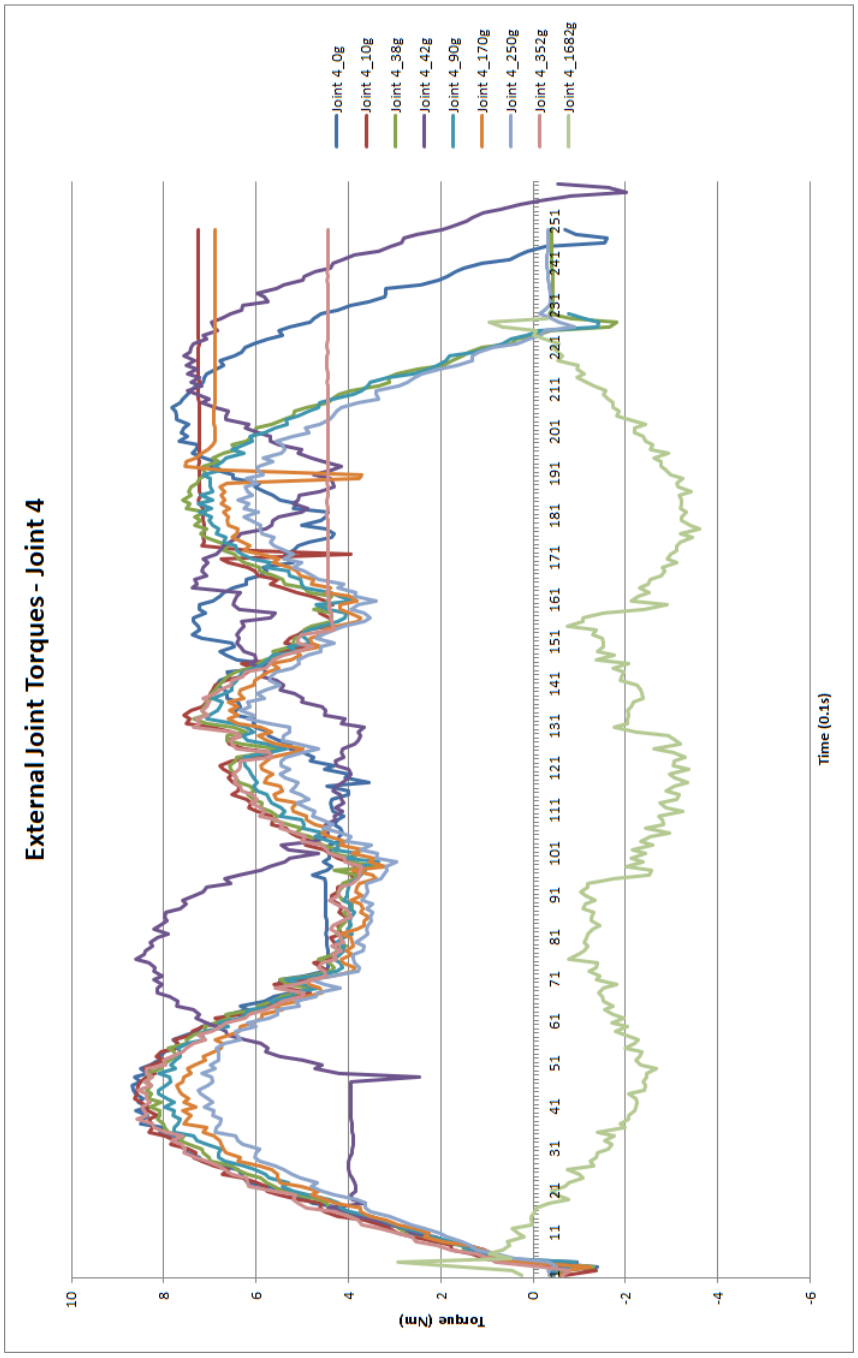



**Figure G.11:** External joint torque at Joint 4

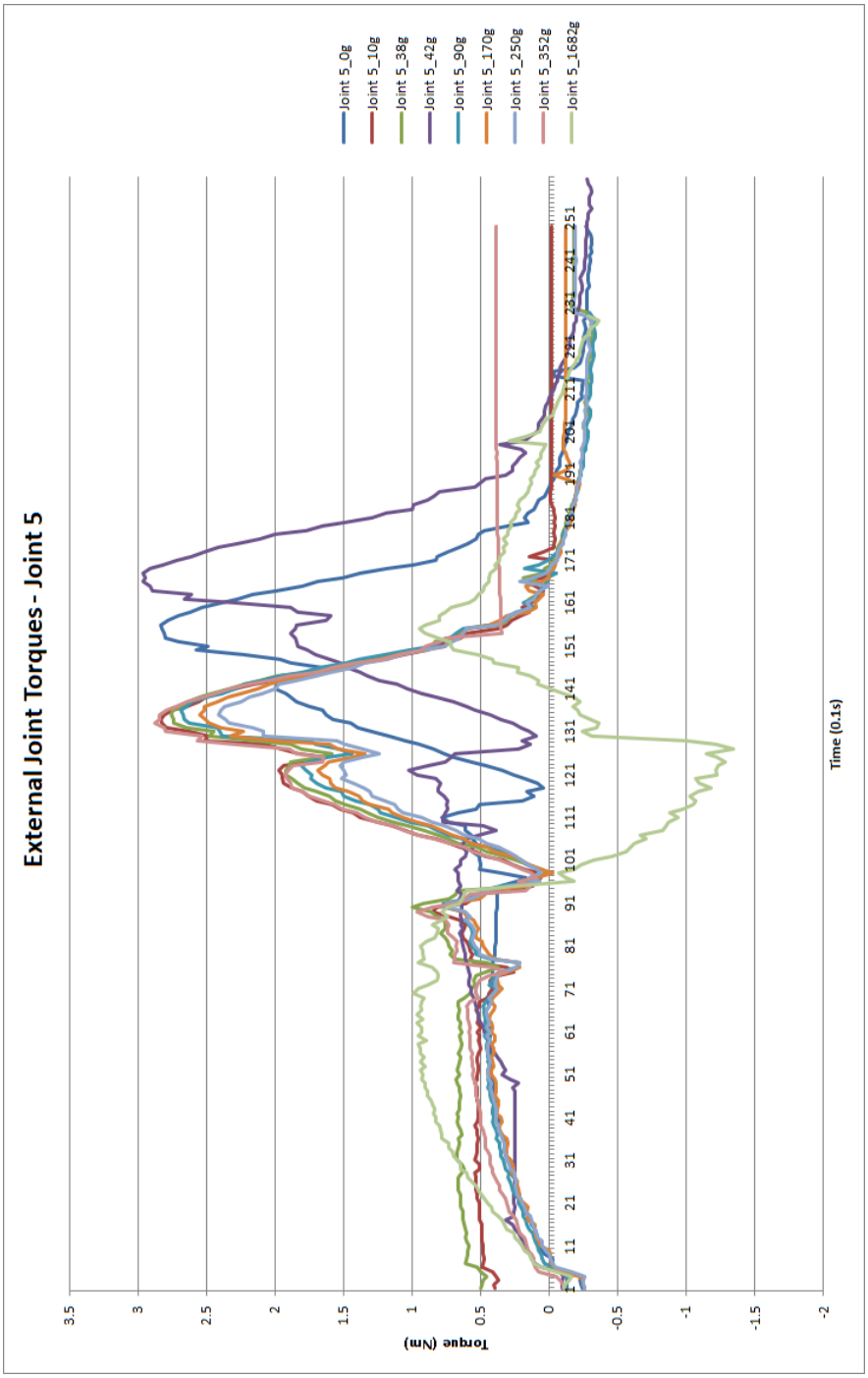



**Figure G.12:** External joint torque at Joint 5

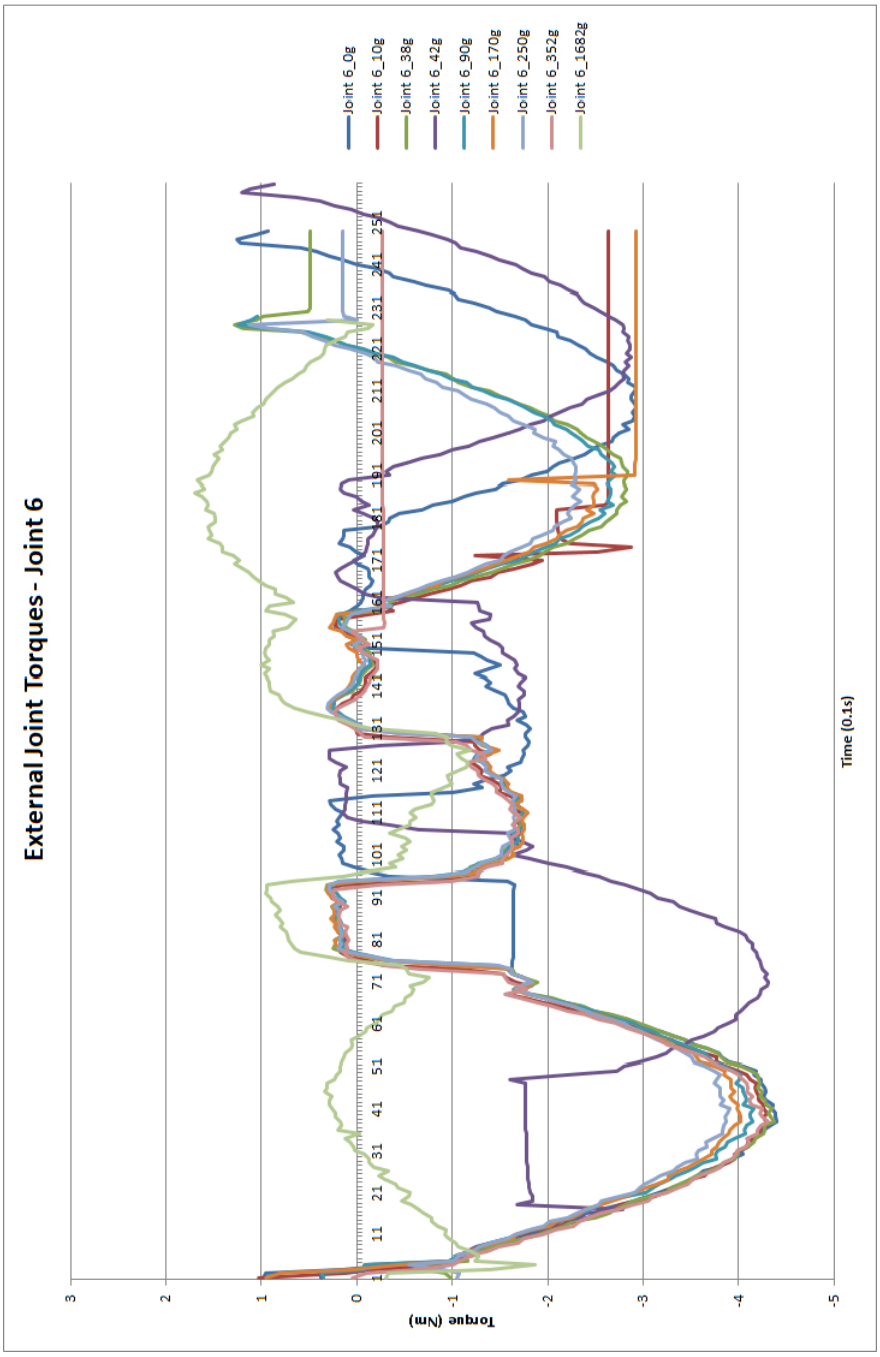



**Figure G.13:** External joint torque at Joint 6

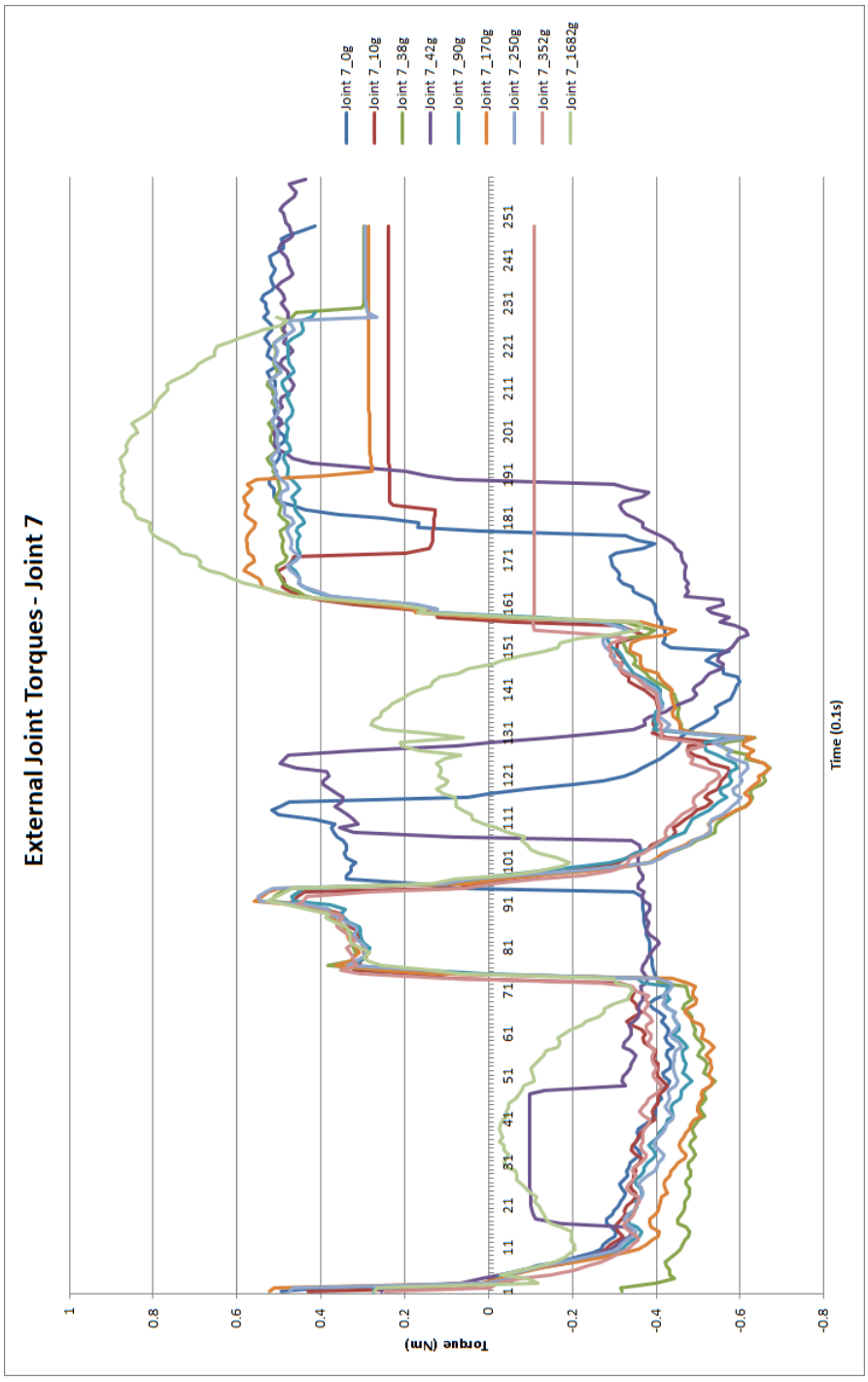



**Figure G.14:** External joint torque at Joint 7

# H

# Appendix

## H.1 Virtual commissioning with SiL method

### H.1.1 Visual Components : Siemens S7 connection procedure [41]

1. Siemens S7 connection plugin can be configured on the connectivity tab on Visual Components.
2. The PLC programming is performed on TIA portal V15 and S7-PLCSIM V15 was used as the virtual controller.
3. TIA portal V15 is launched and a new project is created.
4. The PLC device available is chosen and the project screen is launched.
5. The properties tab of the project is opened by clicking right mouse button.
6. The support simulation during clock compilation option is ticked on.
7. The IP address of the project is filled. IP address is 192.168.0.1 and Subnet mask is 255.255.255.0
8. The process image property is set to none.
9. Protection and security tab is opened and full access option is selected. Permit access with PUT/GET communication from remote partner is chosen.
10. The PLC program is created on the program blocks tab.
11. The tag table with input and output tags are exported to excel sheet.
12. All the applications are closed and NettoPLCSim application is launched.
13. A new server is created with IPV4 as the network IP address and PLCSIM IP address as 192.168.0.1.
14. PLC rack and slot is set as 0/1.
15. The server is started and TIA portal is launched and the PLC program is loaded on to PLCSIM virtual controller.
16. VC is launched and the simulation project is loaded.
17. Connectivity tab is opened and Siemens S7 plugin is selected and a server is added.
18. The IPV4 address is entered. Rack and slot details are provided and connection is tested.
19. Once connected, Load PLC symbols option is selected and the excel sheet is loaded.
20. The input variables are tagged in the simulation to server tab.
21. Output variables are tagged in Server to simulation tab.
22. The server is turned on and connected variables tab is opened.

23. All the windows i.e. VC, TIA Portal and PLCSIM is opened and the logic is tested.
24. This Virtual Commissioning in Software in loop (SiL) method is performed with Visual Components software.