


Zagazig University, Faculty of Engineering,
Mechatronics Program

Implementation of KUKA Robotic Arm in Industrial 3D Intelligent Milling

Deployment of industrial KUKA robot in CNC machining and visual servoing through Kinect interfacing on ROS

Graduation project for the degree Bachelor of Science (B.Sc.)

Submitted to Mechatronics Program, Faculty of Engineering, Zagazig University, Egypt

By

Ahmed Emam
Ahmed Saeed
Donna Mustafa
Dua'a Samir
Hoda Mahmoud
Reeham Mohamed

Supervisors

Asst. Prof. Dr.
Ahmed Hamdy Hassanien
Mechanical Engineering Dept.

Asst. Prof. Dr.Ing.
Mohammed Nour A. Ahmed
Computer and Systems Engineering Dept.

Faculty of Engineering, Zagazig University, Zagazig, Egypt

July, 2017

Graduation Project Report submitted to
Zagazig University, faculty of Engineering, Mechatronics Program, Zagazig, Egypt
in partial fulfillment of the requirements for the degree
Bachelor of Science in Engineering (**B.Sc.**)
©2017

Date of Presentation
15. July 2017

Project Team Members

Ahmed Emam	Ahmed Saeed
Donna Mustafa	Dua'a Samir
Hoda Mahmoud	Reeham Mohamed

Supervisors

Asst. Prof. Dr.Ing. **Mohammed Nour A. Ahmed**
Computer and Systems Engineering Department,
Faculty of Engineering, Zagazig University, Egypt

Asst. Prof. Dr. **Ahmed Hamdy Hassanien**
Mechanical Engineering Department,
Faculty of Engineering, Zagazig University, Egypt

Defense Committee

Prof. Dr. **Nabil H. Mostafa** .....
Mechanical Engineering Dept., Faculty of Engineering, Zagazig University signature

Asst. Prof. Dr. **Mohamed Talaat** .....
Electric Power Engineering Dept., Faculty of Engineering, Zagazig University signature

Asst. Prof. Dr.Ing. **Mohammed Nour A. Ahmed** .....
Computer and Systems Engineering Dept., Faculty of Engineering, Zagazig University signature

Asst. Prof. Dr. **Ahmed Hamdy Hassanien** .....
Mechanical Engineering Dept., Faculty of Engineering, Zagazig University signature

Revision SVN.198.50.20180117

In memory of ***Ahmed Emam***

Abstract

This report describes the first commissioning process, and the deployment of both industrial and research applications on KUKA KR6 R900 sixx at Zagazig University. This project is a Bachelor graduation project submitted to the Mechatronics Program on 2017.

The project work-flow included three major phases: **Commissioning the robot**- starting from base setting and mastering the robot to installing different end-effectors and making complete programs, **Implementation of an industrial application**- through developing post-processing tools to convert any conventional G-Code into KUKA Robot Language (KRL) and use the robot for 2D drawing and 3D milling, and **Implementation of two research applications** related to visual servoing- through using Kinect interfacing on ROS to guide the robot with hand gestures and provide human-safe operating zone where the robot stops when a human approaches, which made possible after developing an API to control the robot directly from any PC.

Acknowledgements

We would like to express our deepest gratitude and appreciation to all those who made it possible to complete this project: our project supervisors- Dr. Mohamed Nour Abdelgawad and Dr. Ahmed Hamdy- for their support and guidance throughout the project, KUKA Roboter GmbH for their amazing robots, Mechanical Engineering department's professors for providing us with help and guidance, and Zagazig University for funding our project.

We would also like to thank all the numerous people in the Internet who ask questions and provide answers for programming problems (specifically in ROS) and for the very useful code and tools they share with others.

Last but not least, we would like to thank our families and friends for always encouraging us onwards. We can never thank them enough for their love and faith.

Contents

List of Figures

List of Tables

"There will be fewer jobs that a robot cannot do better."

— Elon Musk, (Canadian-American business magnate, investor, engineer, and inventor, 1971)

Chapter 1
Introduction

Nowadays any kind of modern production use more and more automatic solutions to achieve new goals. That is why automatic systems use in most cases different robot manipulators. Industries that employ robots in a wide variety of applications are the main customers for robot manufacturers. The manipulator market for research applications, on the other hand, is simply too small for the robot manufacturing industry to develop models specifically for such use. While the hardware and mechanical requirements of developed robots are often similar for both industry and research, scientific software requirements are quite different and even contradictory in many aspects. The goal of scientists is to try to gain as much control over the robot as possible, whereas industries seek safe and easy operational interfaces.

Some of the pioneering companies in the Industrial robotics market are ABB Ltd., Fanuc Corp., Yaskawa Electric Corp., Apex Automation and Robotics, Mistubishi Electric Corp. and KUKA AG. We had the opportunity to work with the latter, KUKA AG, on their model KR6 R900 sixx.

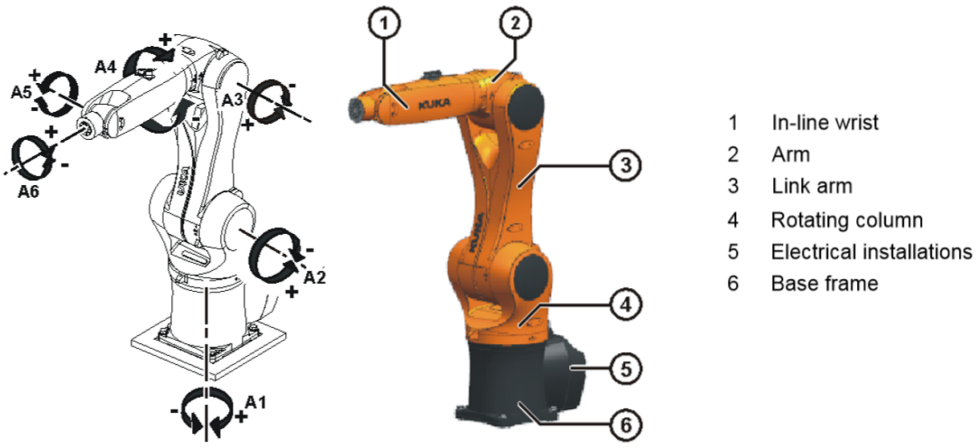


Figure 1.1.: KUKA KR6 R900 sixx robotic manipulator

This robot can carry up to **6**kg of payload, and reach any point in range of **900**mm away from its base with **six** degrees of freedom posing. The robot model name indicates these specifications "KR**6** R**900 sixx**". It's part of the **KR AGILUS** series, which includes many other robot varieties for the payload, maximum reach, and the degrees of freedom.

This robot was purchased three years before starting this project, but no one had commissioned the robot nor installed it properly. We were the first group to use this robot.

Before using any industry level robot arm, it first needed to be fixed of a proper-vibration free base setting, then calibrated to get its encoders to work properly. We needed to go through the manuals to learn how to fix, master (calibrate),and program the robot. The design and manufacturing of a base to support the robot during heavy duty operation included performing mathematical calculations based on the robot's weight and forces to obtain the optimal dimensions and weight for the base, besides performing CAD studies on the manipulator's body to support the results of the mathematical analysis.

We'd done lots of experiments to learn the syntax of the KRL (KUKA Robot Language). Too much time had been consumed due to lack of resources. We'd found that in addition to robots' ability to perform several tasks; by adding the desired end-effector and designing the corresponding programming tool, they offer more variations of one task offered by a conventional machine. We'd decided that our project would include two more phases: **Implementation of an industrial application**- through developing post-processing tools to convert any conventional G-Code into KUKA Robot Language (KRL) and use the robot for 2D drawing and 3D milling, and **Implementation of two research applications** related to visual servoing- through using Kinect interfacing on ROS to guide the robot with hand gestures and provide human-safe operating zone where the robot stops when a human approaches, which made possible after developing an API to control the robot directly from any PC.

We decided to use the robot in milling, because robots offer ease of use through programming, flexibility, speed, precision, cost reduction, repeatability and some of these robots even offer different mounting choices, as they could be mounted on ceilings and walls not just floors.

We first made a 2D-level post-processor extension on Inkscape software, that automatically converts any drawings and text into KRL code files. These files were transferred to the robot, and when executed, the robot made the exact required task. We'd attached a pen, simulating a laser beam to demonstrate this through drawing. Later, we were able to develop 3D post-processing tools for the 3D milling process.

"It is not the beauty of a building you should look at; its the construction of the foundation that will stand the test of time."

— *David Allan Coe*

Chapter 2

Base Settings and CAD analysis

A solid, strong base is required for a robust and free of vibrations motion of the robot. This includes performing mathematical calculations based on the robot's weight and forces to obtain the optimal dimensions and weight for the base, besides performing CAD studies on the manipulator's body to support the results of the mathematical analysis.

After discussing different base settings, a simple design that consists of a cylindrical body, that will support the robot, with upper and lower flanges, was chosen. We then went through designing, simulating, and validating our calculations before manufacturing and installing the base.

2.1. Mathematical Design

Base Structure The base body is mainly constructed of:

1. Two lower flanges:
   - 1st flange is screwed to the ground and held strongly by the concrete layer beneath it.
   - 2nd flange connects the cylindrical body of the base to the previously mentioned flange.
2. The cylindrical body that supports the robot. It's mounted on the 2nd lower flange to stand still while the robot is operated at full speed.
3. The upper flange which holds the robot and is welded to the cylinder.
4. Ribs: there are four upper and lower ribs evenly distributed on the upper and lower flanges, that helps to support the loads exerted on the cylinder.

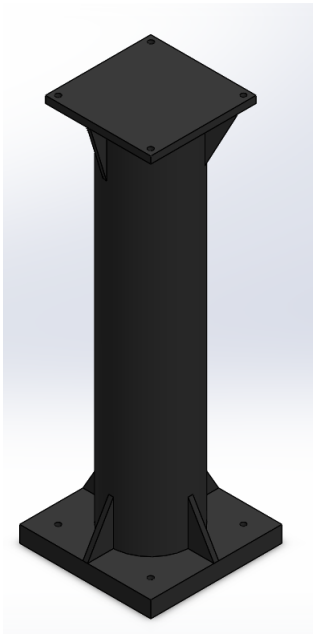


Figure 2.1.: Base Structure

2.1.1. Design of Base Based On Strength

Base Calculations This section discusses the process of calculating the following:

1. Cylinder
   - Inner and outer diameters of the hollow cylinder.
   - The height of the cylinder.
2. Flanges
   - Dimensions (length x width) of the 3 flanges
   - The exact locations of the holes through which the bolts will be screwed
   - Thickness of the flanges
3. Bolts
   - Diameters of all bolts used in the installation and mounting of the base.
4. Ribs
   - Dimensions of the triangular ribs (height x width x thickness)

2.1.1.1. The inner and outer diameters of the cylinder

Since the cylindrical body of the base undergoes different axial and torsional loads, it can be considered as a shaft.

Determining the cylinder diameters First attempt was to design the cylinder using the predefined mechanical properties of the commercial materials available in the market, to get the inner and outer diameters. After several weeks of research and calculation, the results were within the range of 10^{-2} to 10^{-3} mm, which is very small compared to the expected geometry of the cylinder, leaving us to pursue another manner of designing, based on the strength of material.

Design based on strength Second design attempt aimed to use the inner and outer diameters of the cylinders available in the market, to check whether the resulting shear stress would exceed the maximum allowable shear the material can withstand or not. The ASME code for this method is:

$$\tau_{allowable} = \frac{16}{\pi d_o^3(1-k^4)}\sqrt{(k_b M_b + \frac{F_a d_o(1+k^2)}{8})^2 + (k_t M_t)^2} \tag{2.1}$$

where:

- Maximum shear stress the material undergoes upon action of axial and torsional loads: τ_{all}
- Combined shock and fatigue factors: $k_b = k_t = 1:2$ *For safety purposes, the value of k_b and k_t constants is considered 2 in the calculations performed.*
- Bending Moment: $M_b = F_h H$
- Cylinder height: H
- Horizontal shear force acting on the base: F_h
- Torsional Moment: M_t
- Axial force (tension/compression) acting on the base: F_a
- Ratio of the inner to outer diameter: $k = \frac{d_i}{d_o}$
- Maximum Allowable shear stress of a predefined specifications of a material, without keyways:

  $$\tau_{all} = 0.3\sigma_y$$
  $$\tau_{all} = 0.18\sigma_u$$

  These values shall be reduced by 25% in the presence of keyways
- Maximum allowable shear stress of commercial steel, without keyways:

  $$\tau_{all} = 55MPa$$

The key is a machine element, that connects a rotating machine element to the shaft. It's mainly used to prevent relative rotation between the two parts. A keyway is a slot in the shaft and the machine's rotating element, in which the key is seated. In our case, there are no keyways used, because the base is built to be held stationary.

Using the following values of loads acting on the foundation mentioned in the KR AGILUS specification manual:

Type of load	Force / Torque	
	Normal Operation	Maximum load
F_v = Vertical Force	$F_{v\,normal}$ = 967 N	$F_{v\,max}$ = 1297 N
F_h = Horizontal Force	$F_{h\,normal}$ = 1223 N	$F_{h\,max}$ = 1362 N
M_r = Torque	$M_{r\,normal}$ = 367 N	$M_{r\,max}$ = 880 Nm

Table 2.1.: Loads acting on the foundation

Using steel A37 with the following specifications:

- Yield Strength: $\sigma_y = 235MPa$
- Ultimate Tensile Strength: $\sigma_u = 360MPa$

The max allowable shear stress is:

$$\tau_{all} = 0.3(235) = 70.5MPa$$

$$\tau_{all} = 0.18(360) = 64.8MPa$$

The following table shows the different values of diameters available in market, along with the corresponding resulting shear stress:

Outer Diameter (d_o)	Inner Diameter (d_i)	Thickness	Constant k	Shear Stress (τ_{all})
170 mm	150 mm	10 mm	0.88235	29.893 M Pa
220 mm	200 mm	10 mm	0.90909	17.214 M Pa
220 mm	196 mm	12 mm	0.89090	10.898 M Pa

Table 2.2.: Commercially available diameters and their resulting shear stress

Thus, by comparing the previous values of resulting shear stress to the minimum value of the maximum allowable shear stress the material can withstand, 64.8 MPa, we find that any of the alternatives would be suitable for manufacturing the cylinder.

2.1.1.2. Bolts' diameters

- Lower flange

To get the diameters of the used bolts, we need to calculate the tension forces acting on them first.

$$M_b = 2F_1r_1 + 2F_2r_2 = HF_h \tag{2.2}$$

$$\frac{F_1}{r_1} = \frac{F_2}{r_2}$$

$$F_1 = F_2\frac{r_1}{r_2}$$

Substituting F_1 in equation (1.2), we get:

$$HF_h = 2F_2(\frac{r_1^2}{r_2} + r_2) \tag{2.3}$$

Where r_1 and r_2 are the distances from one edge of the flange to the first pair of bolts, and the second pair respectively, as shown in figure below.

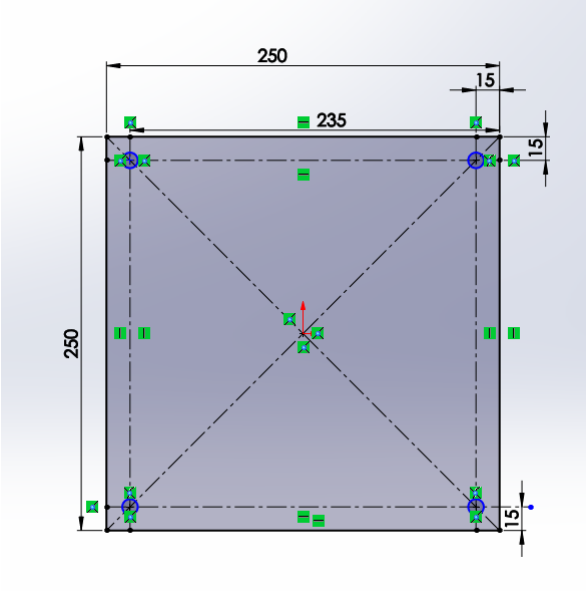


Figure 2.2.: values of r_1 and r_2

Notes:

– The values of dimensions of the flanges and distances r_1 and r_2 are defined based on the assumption that the bolts used in the lower flange is 10 mm, until reasonable results are obtained using try and error.

– All dimensions mentioned in this chapter are measured in mm.

If the diameter of bolts holding the lower flange is 10 mm, and the area of the flanges is (250x250) mm^2, then the values of r_1, r_2 are 15 mm and 235 mm respectively, as shown in the figure below.
Substituting the values of M_b, r_1, and r_2, we get:

$$F_2 = 2308.89N$$

Repeating the previous steps by substituting F_2 with F_2, we get:

$$F_1 = 147.376N$$

For safety purposes, the design of the bolts will be based on the larger tension force, F_2. After obtaining the tensile forces on the bolts, the tensile strength (σ_t) should be calculated to check for safety:

$$\sigma_t = \frac{F_t}{\frac{\pi}{4}d^2} \tag{2.4}$$

For 10 mm bolts diameter, and the previously calculated tensile force, we get:

$$\sigma_t = \frac{2308.89}{\frac{\pi}{4}(0.01)^2} = 29.398MPa$$

Taking in consideration a factor of safety, and checking for the maximum tensile strength the material can undergo before failing:

$$\sigma_t = \frac{\sigma_y}{F.S.} = \frac{235}{3} = 78.33MPa$$

The resulting tensile stress is less than the maximum allowable tensile stress, hence, the previous design is safe.
It was previously mentioned that we used commercial materials available in the market. The manufactured steel sheets are of thickness 15 mm, calculations need to be made to check for safety according the maximum shear and bearing stresses.
The horizontal shear force exerted on the base will be acting on the bolts as well, however, it is negligibly small because it's mostly eliminated by the existence of friction between the two steel flanges it holds. Moreover, the moment resulting from tightening the bolts produces an axial force eliminating the axial load acting on the base.

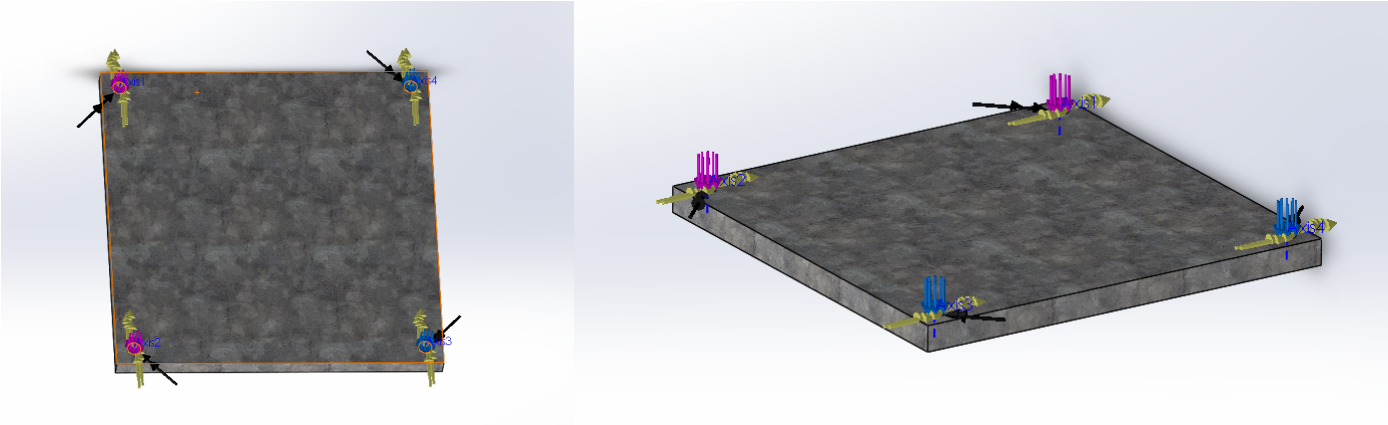


Figure 2.3.: Forces acting on bolts. Black arrows: F_{tr} , purple: F_a, green: F_h

$$F_{total} = \sqrt{(\frac{F_h}{4})^2 + (F_{tr})^2 + 2F_{tr}F_h cos\theta} \tag{2.5}$$

$$F_{tr} = \frac{M_t}{4R} \tag{2.6}$$

Where R,which is the radius of perimeter, has a value of 176.78 mm as shown in figure below.

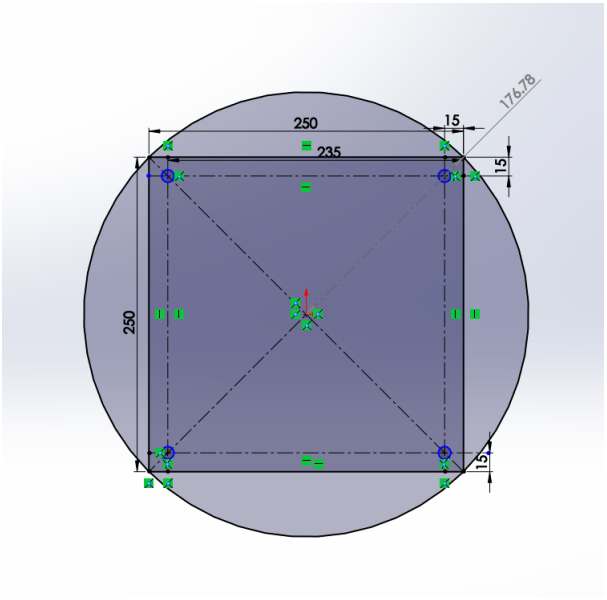


Figure 2.4.: value of R

$$F_{tr} = \frac{880}{4(0.1768)}$$

$$F_{total} = \sqrt{(\frac{1362}{4})^2 + (\frac{880}{4(0.1768)})^2 + (2\frac{1362(880)}{4(4)0.1768}cos45)} = 1504.5N$$

Checking for safety in terms of the bearing stress:

$$\sigma_{br} = \frac{F_{total}}{d_b t} = \frac{1504.5}{0.01(0.015)} = 10.03MPa \tag{2.7}$$

By comparison to the maximum allowable bearing stress, we find that both of the predetermined bolt diameter (10 mm), and the flanges thickness are verified to be safe and suitable, as following:

$$\sigma_{br} = 1.2\frac{\sigma_y}{F.S.} = \frac{1.2(235)}{3} = 94MPa \tag{2.8}$$

Checking on the maximum shear stress exerted on the bolts by the combined forces, we get:

$$\tau = \frac{F_{total}}{\frac{\pi}{4}(d_b)^2} = \frac{1504.5}{\frac{\pi}{4}(0.01)^2} = 19.156MPa \tag{2.9}$$

Comparing it to the maximum shear stress the bolts can stand which is 64.8 MPa since it's made of the steel-37 as well, we can conclude that this is a safe design in terms of the bolts diameters, flanges dimensions, and cylinder diameters.

2.1.1.3. Upper Flange

The upper flange holds the robot base motor, using the bolts provided with the robot. The flange was designed to have an area of (230x230) mm^2, with the locations of 4 bolts holding the robot taken from its base.

2.1.1.4. Ribs

The ribs are made of the same sheet of which the flanges are made of, with a thickness of the ribs is 10 mm.
There isn't an exact way of calculating the height and width of the triangular ribs. After discussing the matter with several machine design doctors, we concluded that the ribs' width should cover the length from the cylinder surface to the edge of flange, resulting in a width of 55 mm, while the height is 100 mm.

2.2. CAD Analysis and Simulation

For the purpose of our study, SolidWorks was used as a CAD software, as it contains solid modeling, motion studies, Simulation PhotoView 360, e-drawing and many other features that were used to obtain a complete CAD model for KR6 r900 sixx KUKA arm. SolidWorks simulation is used to make a stress and strain analysis on designed base model to have a verification on the calculated dimensions of the base before starting fabricating it. Static study is used to perform this analysis as we use maximum loads acting on foundation base

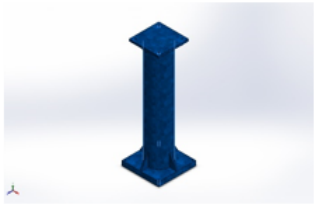
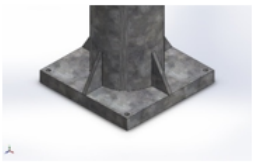
Model Reference	Properties		Components
	**Name:**	Galvanized Steel	SolidBody 1(Cut-Extrude3)(Base)
	Model type:	Linear Elastic Isotropic	
	Default failure criterion:	Max von Mises Stress	
	Yield strength:	2.03943e+008 N/m^2	
	Tensile strength:	3.56901e+008 N/m^2	
	Elastic modulus:	2e+011 N/m^2	
	Poisson's ratio:	0.29	
	Mass density:	7870 kg/m^3	
Curve Data:N/A			

Table 2.3.: SolidWorks definition for the material mechanical properties.

Fixture name	Fixture Image	Fixture Details	
Fixed-1		**Entities:**	1 face(s)
		Type:	Fixed Geometry

Resultant Forces

Components	X	Y	Z	Resultant
Reaction Force(N)	1361.24	1297	-9.45893	1880.23
Reaction Moment (N.m)	0	0	0	0

Table 2.4.: Solidworks model reactions acting on the flange in x, y, and z.

Mesh type	Solid Mesh
Mesher Used:	Standard mesh
Automatic Transition:	Off
Include Mesh Auto Loops:	Off
Jacobian points	4 Points
Element Size	42.7962 mm
Tolerance	2.13981 mm
Mesh Quality Plot	High
Total Nodes	8014
Total Elements	4135

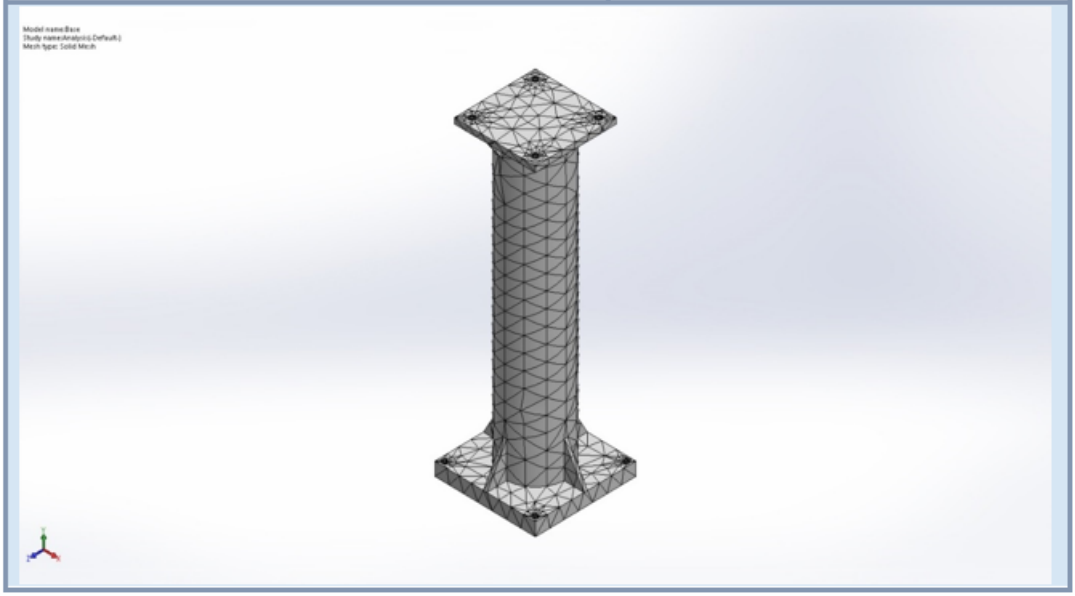


Table 2.5.: Mesh information obtained from Solidworks

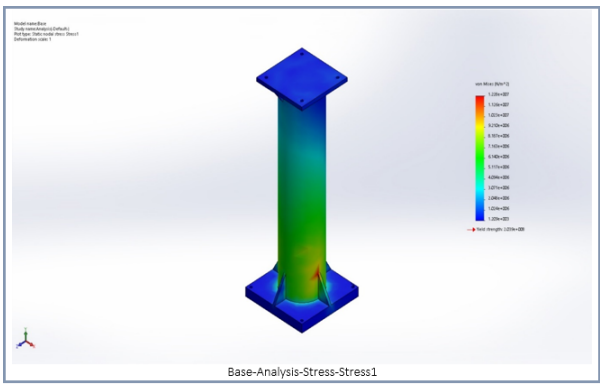


Figure 2.5.: Solidworks representation for stresses acting on the base.

The maximum stress is 10.34 N/m^2 and the minimum stress is 6.8 N/m^2 .

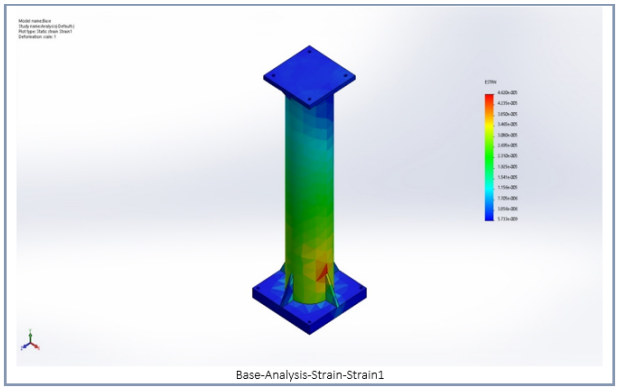


Figure 2.6.: Solidworks representation for the base strain.

The maximum strain is 7.56 and the minimum strain is 6.58.

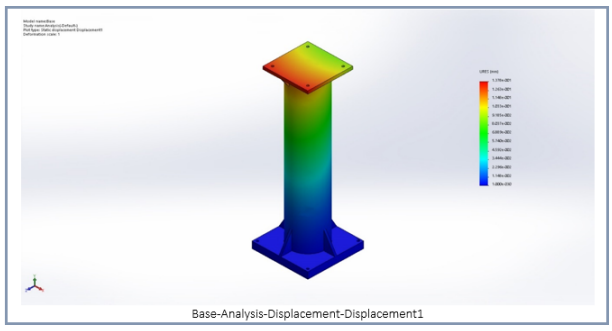


Figure 2.7.: Solidworks representation for displacement at maximum loading.

Displacement of the base due to this forces is 2.7mm at maximum.

2.3. Validation of Design

In simulation process the model can be tested against parameters like static and dynamic response. Instead of using the forces mentioned in the manuals, we modified a CAD model for KR6 r900 sixx to include all the effective masses, then performed a motion analysis study on the model when it's fixed on the base.

2.3.1. Making a Complete CAD Model

2.3.1.1. Searching for a suitable model

All CAD models for KR6 r900 sixx on KUKA website or GrabCAD were step imported parts which are treated as a one body where joints can't rotate therefore, motion study can't be performed because it was impossible to add motors at the robot joints. The solution for this issue was obtained by converting step parts into assembly, which is done through several steps:

- Open the .stp file part in SOLIDWORKS. Select the file type to be .stp
- Click the OPTIONS tab, select Import multiple bodies as parts and click OK.
- Then click Open.

SolidWorks will create an assembly and create an individual part file for each multibody (Part1,Part2,Part3 etc.)

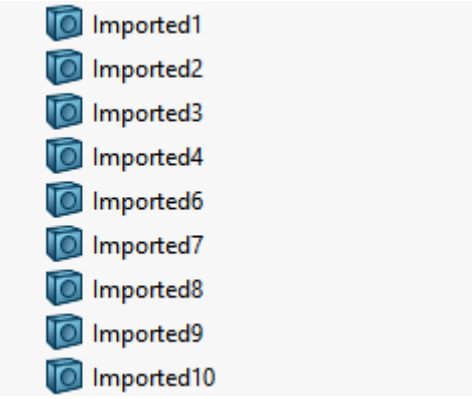


Figure 2.8.: Step parts

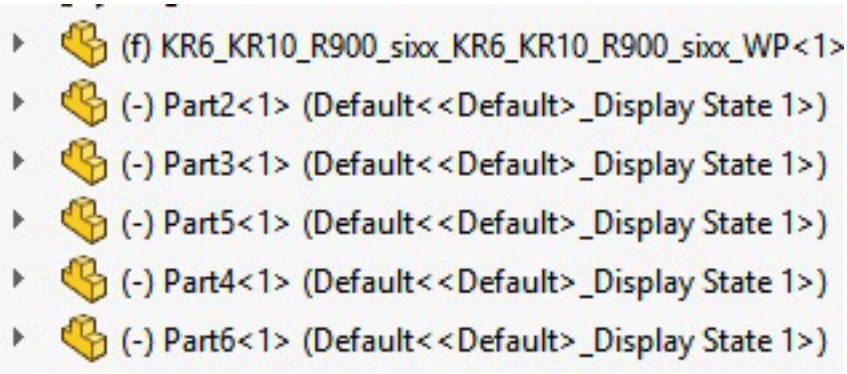


Figure 2.9.: SolidWorks parts

2.3.1.2. Modifications on CAD model

Material and weights The robot material wasn't specified and the model was a whole body, and the total mass for the robot was 33.74 Kg, which was not accurate, because the actual mass of the robot, according to the KR6 R900 sixx dimensions manual, must be approximately equal to 52Kg. This is achieved by making the model hollow, using the shell feature and adding to joins point masses similar to the real motors with mass approximate equal to motors masses.

Figure 2.10.: KUKA motors

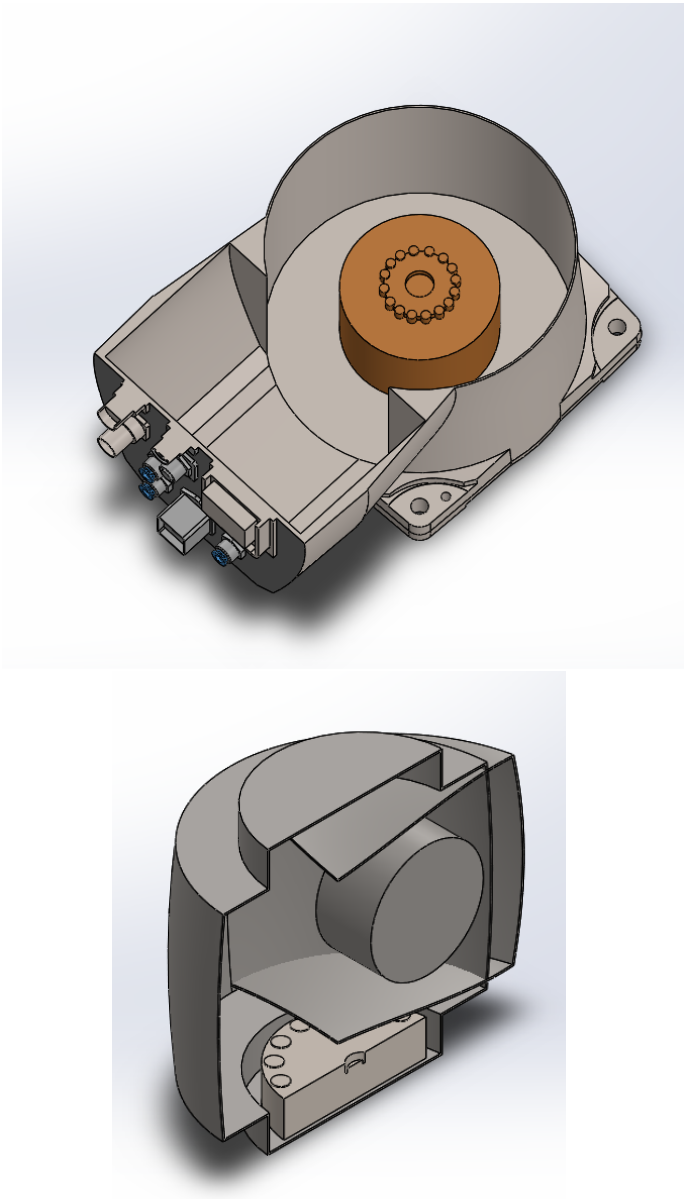


Figure 2.11.: Motors model: Motors 1,2 and 3 model and Motors 4,5 and 6 model

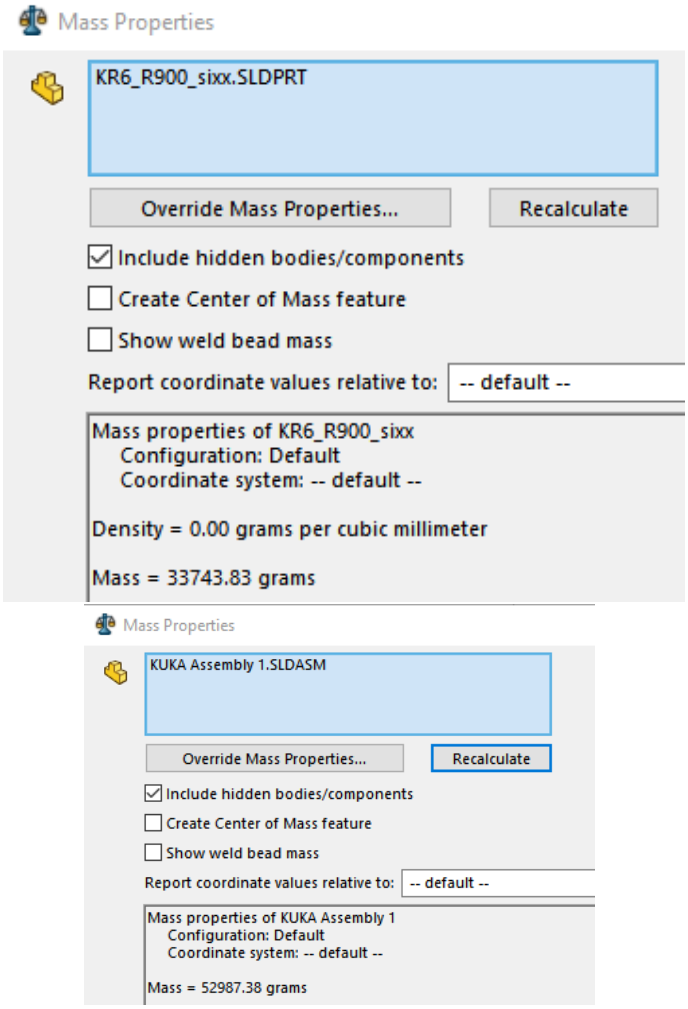


Figure 2.12.: The model mass: Old mass and final mass

Spindle In order to have a complete model for our graduation project, a router spindle and its holder were sketched to complete the existing KUKA model.

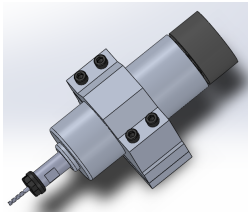


Figure 2.13.: Spindle model

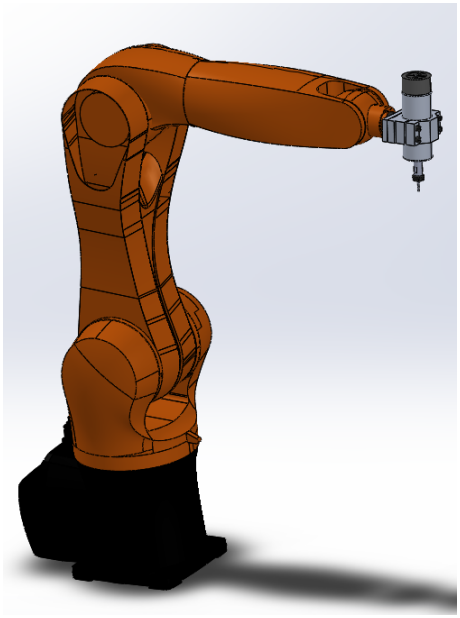


Figure 2.14.: Final CAD model

2.3.2. Motion studies

Motion studies are graphical simulations of motion for assembly models. They simulate and animate the prescribed motion for a model. SolidWorks offer three different types of motion study, Animation, Basic Motion and Motion Analysis. They also offer mate controller that show, save the positions of assembly components at various mate values and degrees of freedom and create simple animations between those positions.

Animation can be used to animate the motion of assembly. If you simply wish to create some nice visuals for presentation or marketing without consideration of mass and gravity effects, then animation is for you.
Basic Motion is an extra layer of complexity that takes into consideration the effects of mass, motors, springs, contact, and gravity on assemblies.

Motion Analysis is the top tier of motion study provides accurately simulation and analyze the motion of an assembly while incorporating the effects of Motion Study elements (including forces, springs, dampers, and friction). Motion Analysis can also be used to plot simulation results for further analysis.

Therefore, Motion Analysis is used to simulate the model, generating results and plots of the simulation and Animation is used to make a video for motion.

2.3.2.1. Animation study

Animation study is done to make an animation video of the moving parts with the use of limit mates. Limit angle mate is an advanced mate type that is performed by

selecting two planes which rotate with respect to each other giving the desired range to mate and input a maximum and minimum value for the angle to specify the desired range of motion. Another advantage of limit angle is that it prevent collision between the moving parts.

Axis	Range of motion, software-limited	Speed with rated payload
1	+/-170°	360 °/s
2	+45° to -190°	300 °/s
3	+156° to -120°	360 °/s
4	+/-185°	381 °/s
5	+/-120°	388 °/s
6	+/-350°	615 °/s

Table 2.6.: Axes range of motion for each robot joint

2.3.2.2. Motion analysis

On implementing motion analysis by adding a motor at the required location to start simulating the robot, a problem arose; the model exploded on running the simulation where some of the parts were separated from each other. This happened because of redundancy constrains. For Motion Analysis studies, having redundant mates is the equivalent of over-defining the model.

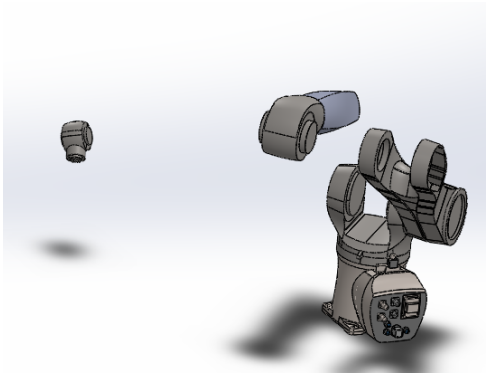


Figure 2.15.: Redundancy constraint problem

This issue was solved by using mechanical mates of hinge type instead of standard mates. Hinge mate limits the movement between two components to one rotational degree of freedom. It has the same effect as adding a concentric mate plus a coincident

mate and also limit the angular movement between the two components by adding limit angles. Reducing the negative effect of redundant mates on analysis.

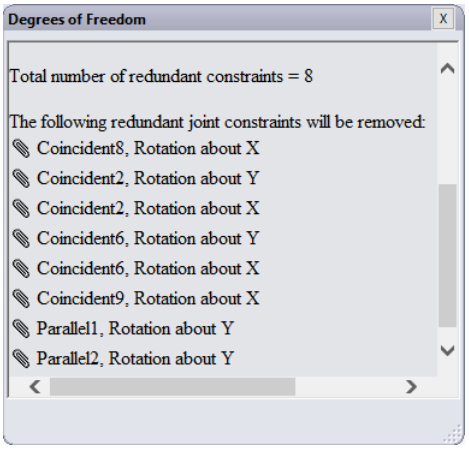


Figure 2.16.: Presence of redundancy constrains

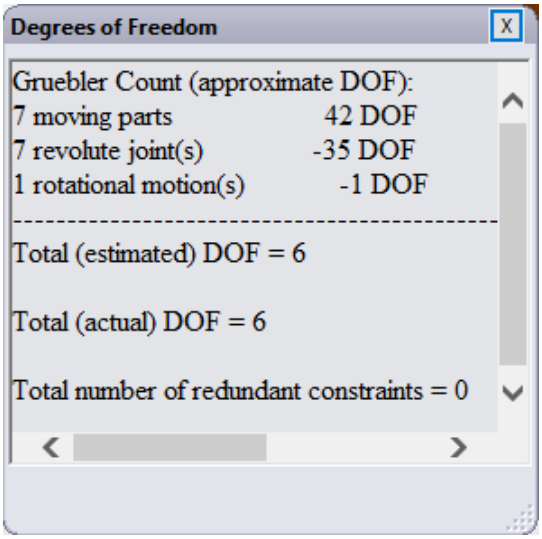


Figure 2.17.: Zero redundant constraints

Now motors can be added to the model to perform any motion and results as motor torque, velocity, acceleration and force are generated.

Results of motion analysis study] By assigning a motor to simulate the motion of axis and plotting the results to achieve this motion we found:

1. Results of motor torque for axis 1 to move 100 degrees in 4 seconds:

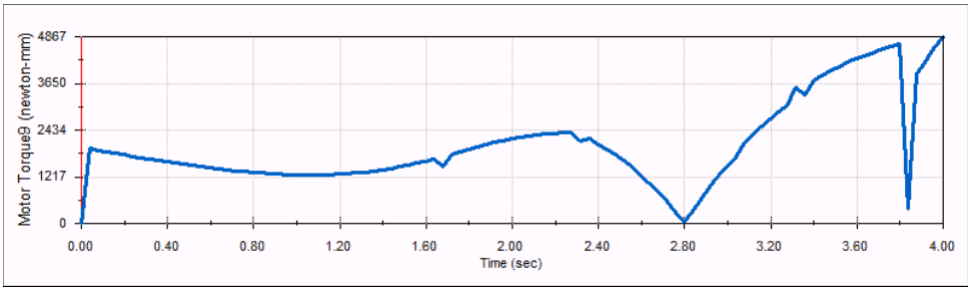


Figure 2.18.: Motor 1 torque

Knowing that the actual motor torque for axis 1 is 4.5 N.m so the motion analysis result is approximate to actual torque.
Motor torque vary with time because of the motion of the links beyond the motor which has an effect on the torque by changing the loads carried by the motor.

2. Motor 2 torque to move 60 degrees in 2 seconds:

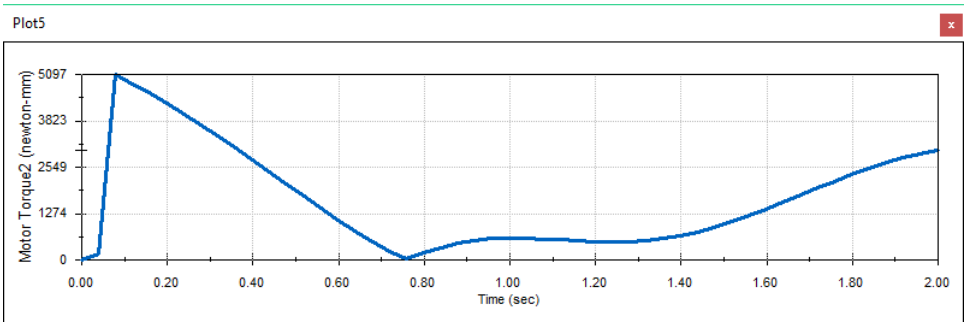


Figure 2.19.: Motor 2 torque

Where the actual motor torque is 4 N.m

3. Motor torque for axis 3 to move 75 degrees in 1.8 seconds:

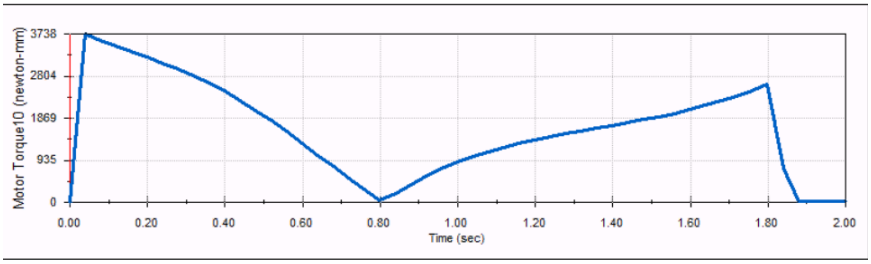


Figure 2.20.: Motor 3 torque

Knowing that the actual motor torque for axis 3 is 3.5 N.m so the motion analysis result is approximate to actual torque.

Trace path Motion analysis results and plots have a trace path option that can trace the path of a point in the assembly. The selected point is end mill vertex to create the curve feature that represent the motion of the links. By assigning a data point motors to the joints whose data is imported from excel spreadsheet of file type .csv containing two columns, first represent time in seconds while other is degrees of rotation. The generated curve of adding this data to joints 1-5 is an arc shape.

	A	B
1	0	0
2	1	10
3	2	20
4	3	30
5	4	40
6	5	50
7	6	60
8	7	70
9	8	80
10	9	90

Figure 2.21.: Used data point

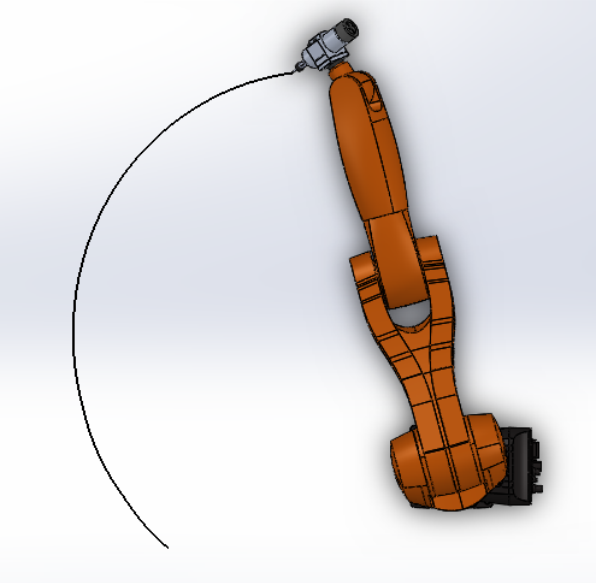


Figure 2.22.: Created curve

2.3.3. Base Manufacturing

Several materials can be used to manufacture the body of the base, such as AISI 1010 & AISI 1010 carbon steel, SAE 304 stainless steel, etc., all of which the mechanical properties are predefined. However, the available material in the market was steel 37, which has the following mechanical specifications:

– Yield Strength: $\sigma_y = 235MPa$

– Ultimate Tensile Strength: $\sigma_u = 360MPa$

The calculations performed on this material are proven to be safe and suitable to our designed base geometry.
The squared, upper and lower flanges are made of a 15mm thick steel-37 sheet, and holes were drilled in the designated locations shown in the sketches below. The flanges are then welded to the cylindrical body of the base. It's worth mentioning that the dimensions of the manufactured base are larger than those calculated, this is due to different manufacturing and safety purposes, the final dimensions manufactured are illustrated below.

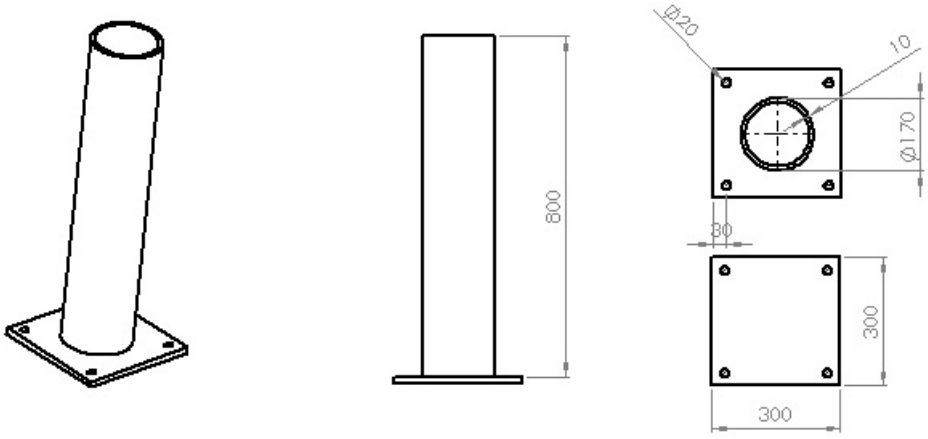


Figure 2.23.: Final dimensions for the base.

For the ribs to be welded, the base needs to be cleaned of any accumulated rust, it is then taken to undergo different machining steps, using a lathe machine. Next, it is painted in black to match the robot colors.
Figure1.4 illustrates different stages of manufacturing.

Figure 2.24.: Manufacturing and Machining of the base

2.3.4. Base Installation

The following steps clarifies the process of installing the robot base:

1. The exact location of the base is chosen according to the safe operating range of the robot, mentioned in the KR AGILUS sixx specification manual.
2. Remove the ceramic plate. The number of plates removed depends on the area of the base and the ceramic plate itself, in our case, only one plate was removed.
3. Scrape the sand and cement layers beneath the ceramic until you reach the concrete, it can be recognized when a layer of gravels appears.
4. Drill the holes, in which the dowel rods will be placed, in the predesigned positions in the concrete. For this design, seven holes were drilled, however, the number of dowels depends on the design.
5. Make a mixture of sand, gravels, cement and anabond adhesive agent to mold a new concrete layer, and keep mixing until it's consistent.
6. Pour in the concrete mixture on top of the pre-existent one, while the dowels are placed, and wait for 2 days to ensure that it's completely solidified.
7. Remove the dowels and start screwing the bolts.
8. Start mounting the first flange. Use a bubble level to check if it lies in a perfectly horizontal position, if not, you must scrape the newly molded concrete layer until it's even.
   Note: Even out the newly molded concrete once it dries, this would help you to avoid such issues in further steps.
9. Mount the base, then move the robot carefully until it's placed on it in the right orientation.

Figure 2.25.: Different stages of the Base installation process

Figure 2.26.: The final installation

Chapter 3
KUKA conditioning

3.1. Robot Mastering

The Mastering operation calibrates the relationship between the position sensor, attached to each axis motor, and each axis angle defined for each robot. Mastering axes enables the definition of geometric parameters used to describe the analytic parameters of a robot's geometric model. This helps in increasing the accuracy of the robot and correcting for discrepancies between design parameters and actual values.
Mastering the robot is performed by moving the each axis into a defined mechanical position, which is known as the "Mechanical zero position". The zero position, which is defined by a reference notch, is an assignment to the axis drive angle. Whenever the robot moves from the mechanical zero position, its deflection represents the change in corresponding axis angle (0 increments for 0 degrees).
To locate the mechanical zero position of a robot axis precisely, the axes must be aligned to their pre–mastering position. The protective cap of the gauge cartridge is then removed and a dial gauge, or the supplied EMD, is fitted to it.

Note: The robot must be mastered in the same temperature conditions (either always cold or at operating temperature) to avoid inaccuracies.

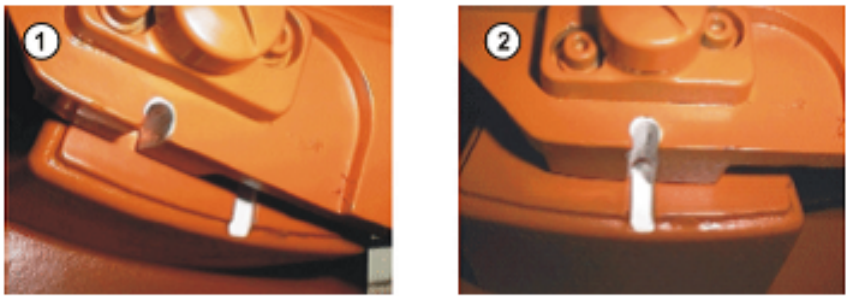


Figure 3.1.: Moving an axis to pre-mastering position

On passing over the reference notch, the gauge pin reaches its lowest point, the mechanical zero position is reached. The electronic measuring tool sends an electronic signal to the controller.

Mastering can be performed through several methods; for older Robot versions it is performed using EMT, as for the KUKA AGILUS, mastering is done using one of these methods; EMD, dial gauge or MEMD. For our purpose, we used the MEMD supplied with the robot. The mastering positions are similar, but not identical, for all robots. Exact positions may vary between individual robots or single robot type.

Note: The robot must be mastered in the same temperature conditions (either always cold or at operating temperature) to avoid inaccuracies.

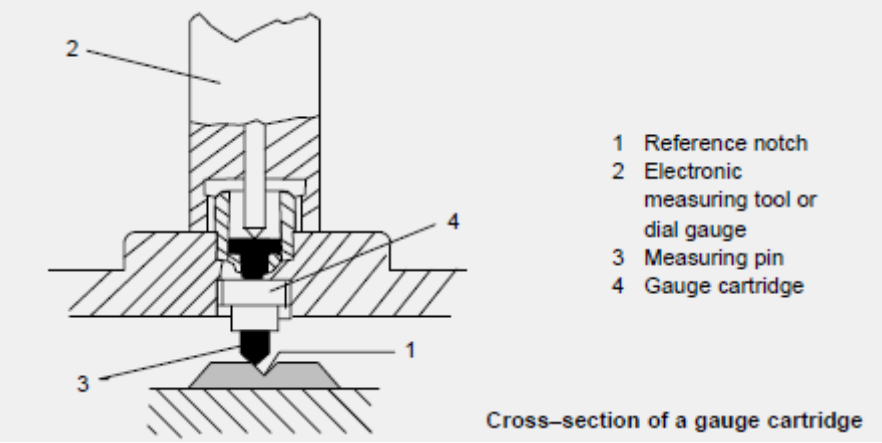


Figure 3.2.: Moving an axis to pre-mastering position

3.1.1. Mastering using MEMD

Unlike Dial gauge mastering, which requires moving the robot manually to the mastering position, MEMD mastering offers automatic movement, done by the robot, to reach the mastering position. Mastering is performed first without a load then repeated using a load. The MEMD mastering tools are shown in the below picture.

Note: The robot must be mastered in the same temperature conditions (either always cold or at operating temperature) to avoid inaccuracies.

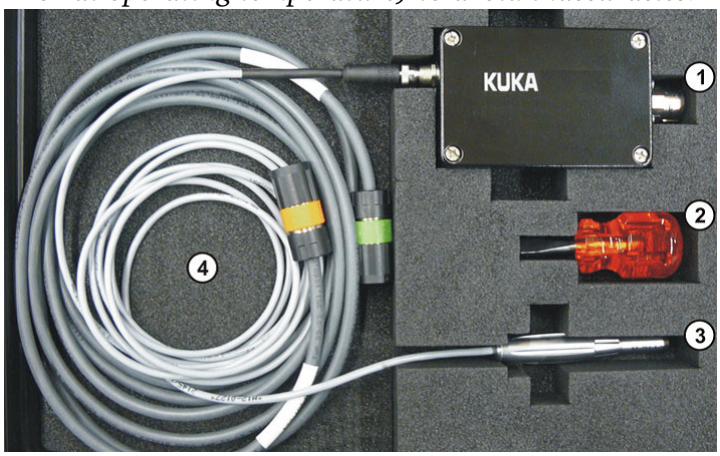


Figure 3.3.: MEMD kit: 1. MEMD box, 2. Screwdriver, 3. MEMD, 4. Cables

Types of mastering

1. First mastering (without a load).
2. Tech offset (with a load and with saving the difference from first mastering being saved).
3. Master load with offset is based on saving an offset value that can be used to calculate first mastering in case it was lost (used when required, carried out with a load for which an offset has already been taught. This type is used to check first mastering or to restore it in case it was lost).

Precondition

- There is no load on the robot; i.e. there is no tool, workpiece or supplementary load mounted.
- A1 to A5 are in the pre-mastering position.
- No program is selected.
- Operating mode T1

Procedure

1. In the main menu, select Start-up > Master > EMD > With load correction > First mastering. A window opens. All axes to be mastered are displayed. The axis with the lowest number is highlighted.
2. Remove the cover from connection X32.
3. Connect the EtherCAT cable to X32 and to the MEMD box.

4. Remove the protective cap of the gauge cartridge on the axis highlighted in the window.
5. Screw the MEMD onto the gauge cartridge.
6. Press Master.
7. Press an enabling switch and the Start key.
8. When the MEMD has passed through the reference notch, the mastering position is calculated. The robot stops automatically. The values are saved. The axis is no longer displayed in the window.
9. Remove the MEMD from the gauge cartridge and replace the protective cap.
10. Repeat steps 4 to 8

Mastering of A6

1. Move A6 to the mastering position. A6 has very fine marks in the metal. Align these marks exactly with one another.
2. In the main menu, select Start-up > Master > Reference.
3. The option window Reference mastering is opened. A6 is displayed and is selected.
4. Press Master. A6 is mastered and removed from the option window.
5. Close the window.
6. Disconnect the EtherCAT cable from X32 and the MEMD box.

For more information about the remaining mastering types (teach offset and mastering load with offset) and other mastering methods (using dial gauge and EMD), please refer to section “5.9 Mastering” in the provided manual “07-KSS_82_Software programming_en”.

3.2. Robot Calibration

Robot calibration is defined as identifying certain parameters in the robot's kinematic structure, as an example; identifying relative position of robot links. Robot calibration can be performed through various methods, two of which are using a predefined and built-in calibration programs, or external methods (hardware and/or software) as RoboDK or advintec TCP. Calibration process differs in complexity from one method to another.

Calibration can be divided into three levels, depending on the type of modeled errors. The first of which models the differences between the actual and reported joint displacement values. This is also known as mastering. The second level, kinematic calibration, is related to the geometry of the robot and performing full geometric calibration, including angle offsets and joint lengths. The third level, non-kinematic calibration, models errors such as stiffness and friction.

Calibration offers higher positioning accuracy for offline programmed robots. Accuracy means that the real position of the robot end effector corresponds better to the actual position calculated from the robot's mathematical model. In the case of offline programming, pose accuracy is considered an important performance criteria.

The calibration method used in out project is the first; using a predefined and built-in calibration program, which can be performed through different procedures in the KUKA platform, varying for tool and base calibration. For base calibration, these procedures are 3-point method, indirect method and Numeric input, and for tool calibration the procedures are XYZ 4-point method and XYZ reference method, both for TCP, ABC 2-point method and Numeric input. For the purpose of our project, the applied calibration procedures for both tool and base calibration were XYZ 4-point method and 3-point method respectively.

3.2.1. Tool calibration using XYZ 4-point procedure

The TCP of the tool to be calibrated is moved to a reference point from four different directions. The reference point can be freely selected. The robot controller calculates the TCP from the different flange positions. These four directions must be sufficiently different from one another (similar to the positions shown in the provided pictures).

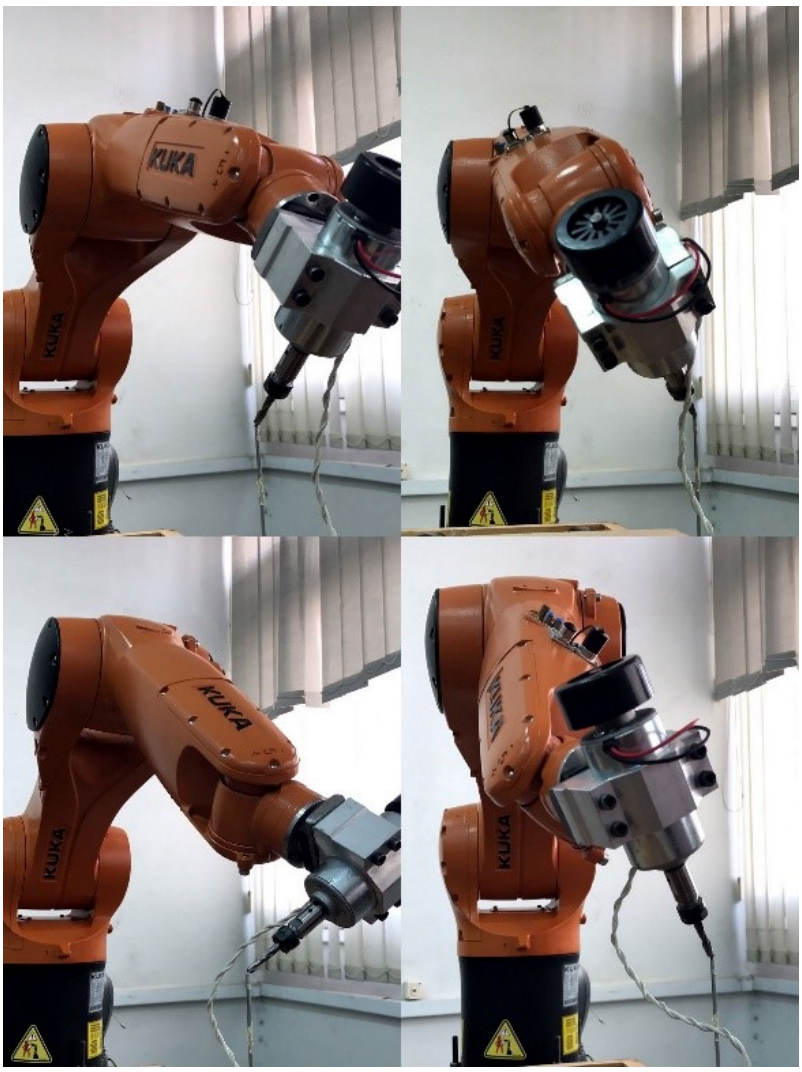


Figure 3.4.: Tool calibration using XYZ 4-point procedure

Precondition

- A previously calibrated tool is mounted on the mounting flange.
- Operating mode T1

Preparation

Calculate the TCP data of the calibrated tool:

1. In the main menu, select Start-up > Calibrate > Tool > XYZ Reference.
2. Enter the number of the calibrated tool.
3. The tool data are displayed. Note the X, Y and Z values.
4. Close the window.

Procedure

1. In the main menu, select Start-up > Calibrate > Tool > XYZ Reference.
2. Assign a number and a name for the new tool. Confirm with Next.
3. Enter the TCP data of the calibrated tool. Confirm with Next.
4. Move the TCP to a reference point. Press Calibrate. Answer the request for confirmation with Yes.
5. Move the tool away and remove it. Mount the new tool.
6. Move the TCP of the new tool to the reference point. Press Calibrate. Answer the request for confirmation with Yes.
7. Enter the payload data. (This step can be skipped if the payload data are entered separately instead.) (»> 5.12.3 "Entering payload data" Page 138)
8. Confirm with Next.
9. If required, coordinates and orientation of the calibrated points can be displayed in increments and degrees (relative to the FLANGE coordinate system).
10. For this, press Meas. points. Then return to the previous view by pressing Back.
11. Either: press Save and then close the window via the Close icon. Or: press ABC 2-point or ABC World. The previous data are automatically saved and a window is opened in which the orientation of the TOOL coordinate system can be defined.

3.2.2. Base calibration using 3-point method

During base calibration, the user assigns a Cartesian coordinate system (BASE coordinate system) to a work surface or the workpiece. The BASE coordinate system has its origin at a user-defined point. In 3-point calibration, the robot moves to the origin and 2 further points of the new base. These 3 points define the new base.

Advantages of base calibration

1. The TCP can be jogged along the edges of the work surface or workpiece.
2. Points can be taught relative to the base. If it is necessary to offset the base, e.g. because the work surface has been offset, the points move with it and do not need to be retaught.

Precondition

- A previously calibrated tool is mounted on the mounting flange.
- Operating mode T1

Procedure

1. In the main menu, select Start-up > Calibrate > Base > ABC 3-point.

2. Assign a number and a name for the base. Confirm with Next.
3. Enter the number of the mounted tool. Confirm with Next.
4. Move the TCP to the origin of the new base. Press Calibrate. Answer the request for confirmation with Yes.
5. Move the TCP to a point on the positive X-axis of the new base. Press Calibrate. Answer the request for confirmation with Yes.
6. Move the TCP to a point in the XY plane with a positive Y value. Press Calibrate. Answer the request for confirmation with Yes.
7. If required, coordinates and orientation of the calibrated points can be displayed in increments and degrees (relative to the FLANGE coordinate system). For this, press Meas. points. Then return to the previous view by pressing Back.
8. Press Save.

Note

For more information about tool and base calibration, please refer to section “5.11 Calibration” in the provided manual “07-KSS_82_Software programming_en”.

3.3. WorkVisual and LAN connection

The WorkVisual software package is the engineering environment for KR C4 controlled robotic cells. It offers the following functionalities:

- Configuring and connecting field buses
- Programming robots offline
- Configuring machine data
- Configuring machine data
- Editing the safety configuration
- Transferring projects to the robot controller
- Loading projects from the robot controller
- Comparing a project with another project and accepting differences where necessary
- Managing long texts
- Managing option packages
- Diagnostic functionality
- Online display of system information about the robot controller
- Configuring traces, starting recordings, evaluating traces (with the oscilloscope)

Hardware requirements: Minimum requirements

- PC with Pentium IV processor, min. 1500 MHz
- 512 MB RAM
- DirectX8-compatible graphics card with a resolution of 1024x768 pixels

Recommended specifications

- PC with Pentium IV processor and 2500 MHz
- 1 GB RAM
- DirectX8-compatible graphics card with a resolution of 1280x1024 pixels

Software requirements:

- Windows 7 (Both the 32-bit version and the 64-bit version can be used).
- Or: Windows XP (32-bit version, with at least Service Pack 3, the 64-bit version cannot be used).

If the following software are not already installed on the PC, the installation wizard automatically starts their installation before preceding with the WorkVisual installation.

- .NET Framework 2.0, 3.0 and 3.5
- SQL Server Compact 3.5
- Visual C++ Runtime Libraries
- WinPcap

3.3.1. WorkVisual Installation

1. Start the program setup.exe.
2. If the following components are not yet installed on the PC, an installation wizard opens:
   - NET Framework 2.0, 3.0 and 3.5 Follow the instructions in the installation wizard. .NET Framework is installed.
3. If the following component is not yet installed on the PC, an installation wizard opens:
   - SQL Server Compact 3.5 Follow the instructions in the installation wizard. SQL Server Compact 3.5 is installed.
4. If the following components are not yet installed on the PC, an installation wizard opens:
   - Visual C++ Runtime Libraries
   - WinPcap Follow the instructions in the installation wizard. Visual C++ Runtime Libraries and/or WinPcap is installed.
5. The WorkVisual [...] Setup window opens. Click on Next.
6. Accept the license agreement and click on Next.
7. Click on the desired installation type.
8. Click on Install. WorkVisual is installed.
9. Once installation is completed, click on Finish to close the installation wizard.

Note

For more information about installation, uninstallation and GUI of the WorkVisual software, please refer to Manual “KST_WorkVisual_en”.

3.3.2. LAN connection

In order to start file sharing process and to be able to use all functions of WorkVisual, a PC-Controller connection must be established. There are several ways to connect the

KRC4 and KUKA-PC, one of which is setting up a local network for the connection of between several devices. This can be done by either setting static IPs for the connected devices and connecting them physically using a specified Ethernet cable, or by using a network router and assign dynamic IPs starting from a specified value, with specified number of connected devices.

To obtain and/or change IP values for the PC

1. Open "Network and sharing center"
2. Choose "Change adapter settings"
3. Right click "Ethernet connections"
4. Choose "Internet protocol version 4 (TCP/IPv4)"
5. Choose "Properties"
6. Next you either set static IPs or choose dynamic IPs to be set, in our case by the router.

Note

The following address ranges are used by default by the robot controller for internal purposes. IP addresses from this range must not therefore be assigned by the user.

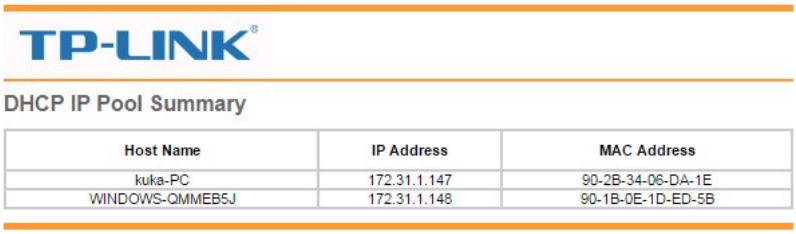
- 192.168.0.0 ... 192.168.0.255
- 172.16.0.0 ... 172.16.255.255
- 172.17.0.0 ... 172.17.255.255

LAN Configuration steps

1. Connect the PC and the KRC4 to the router using regular Ethernet cables.
2. Access the router configuration page using the given data on the back of the router. (the router used in our case is TP-LINK, with username and password both being admin)
3. In the Interface setup change the LAN settings to your preferred values
4. It is preferred to set the starting IP address similar to that of the KRC4 (172.31.147) to avoid conflicts. Network gateway value is (172.31.1.1) and subnet mask (255.255.0.0), all other settings shall remain unchanged.

5. After changing these values, the IP address used to access the router settings will change from (192.168.1.1) to the set gateway value (172.31.1.1) but with the same user name and password.

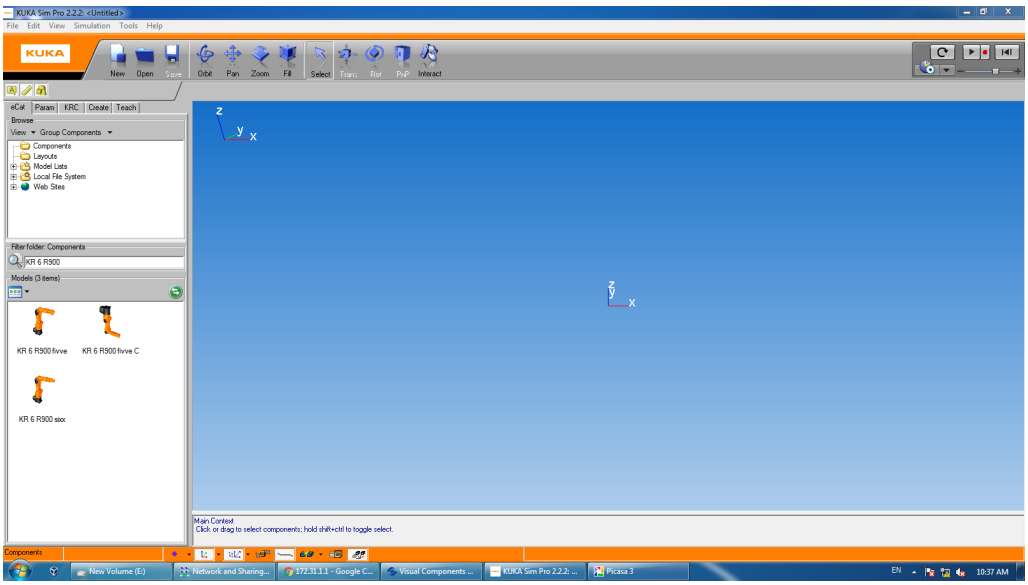
   fig6



6. The connection is now established and can be verified by checking the router LEDs

3.4. Installation of KUKA.Sim Pro

KUKA.Sim Pro is used for the complete offline programming of KUKA robots. This product allows the analysis of cycle times and the generation of robot programs. It also enables a real-time connection to the virtual KUKA robot controller (KUKA.OfficeLite). KUKA.Sim Pro is additionally used for building parametric components and defining kinematic systems, which can also be used in KUKA.Sim Layout and KUKA.Sim Tech. KUKA.OfficeLite is included in the KUKA.Sim Pro package. CAD importers are available as an option. This requires a purchasable license for each import interface.



Requirements

- The minimum requirements for the computer are a 2 GHz CPU and 2 GB RAM, and an OpenGL-capable graphics card with at least 512 MB RAM and a resolution of 1024 x 768 pixels or a similarly specified notebook.
- Supported operating systems are WIN XP - 32-bit or WIN 7 - 32/64-bit.

3.4.1. Installation

1. Start the setup file ".exe"
2. Read the agreement and activate the check box "I accept ..."
3. Select "Install" to start installation.
4. Select "Install" to complete installation

Installing the component library After installation of the KUKA.Sim software product, the component library is installed with the following program: SetupKUKASim-Library_2.2.0_Buildx.exe The procedure is very similar to the installation of the software products. Simply follow the instructions given. The KUKA.Sim component library contains over 1,000 typical layout components (robots, grippers, fences, etc.), various demo layouts and tutorials for KUKA.Sim. Although the KUKA.Sim products still work without the component library installed, it is strongly recommended that the component library is installed in order to be able to create layouts quickly and easily.

3.4.2. License types

There are different types of licensing for the KUKA software. License types are determined and verified in accordance with the purchase made from KUKA Roboter GmbH. The software licensing concerning the KUKA arm at Zagazig university is an educational bundle license. The serial number for the license is found in the booklet of the KUKA.Sim Pro CD. Information about different licensing bundles are obtained by contacting KUKA Roboter GmbH by email **simulation@kuka-roboter.de.** Further details about the steps of obtaining the serial key, for different commercial bundles, are found on page 13 of (KUKA.Sim 2.2- Installation-en) manual. The serial number associated with this purchase is: **K5P22-N174H-AW7KY-9**

3.4.2.1. Stand-alone License

The license is on the PC on which KUKA.Sim is used. The license key is then valid for this PC only. It can also be transferred to a different PC, but cannot be used on a two different PCs at the same time, or when either of the two PCs is off.

3.4.2.2. Network License

Network licenses provide a flexible way of using KUKA.Sim on more than one PC. When a license is requested by a PC, this license is then allocated to this PC. When KUKA.Sim is closed, the license becomes available again and can be accessed by other PCs. A license server is required to manage the network licenses. When KUKA.Sim Pro is started, the computer's identity (IP address, please refer to **LAN connection** in manual Section "WorkVisual & LAN connection") is required occasionally, however, KUKA.Sim Pro needs to check with the local license server to make sure that KUKA.Sim Pro and server are on the same PC, which is required in the network license configuration.

Requesting a license file manually

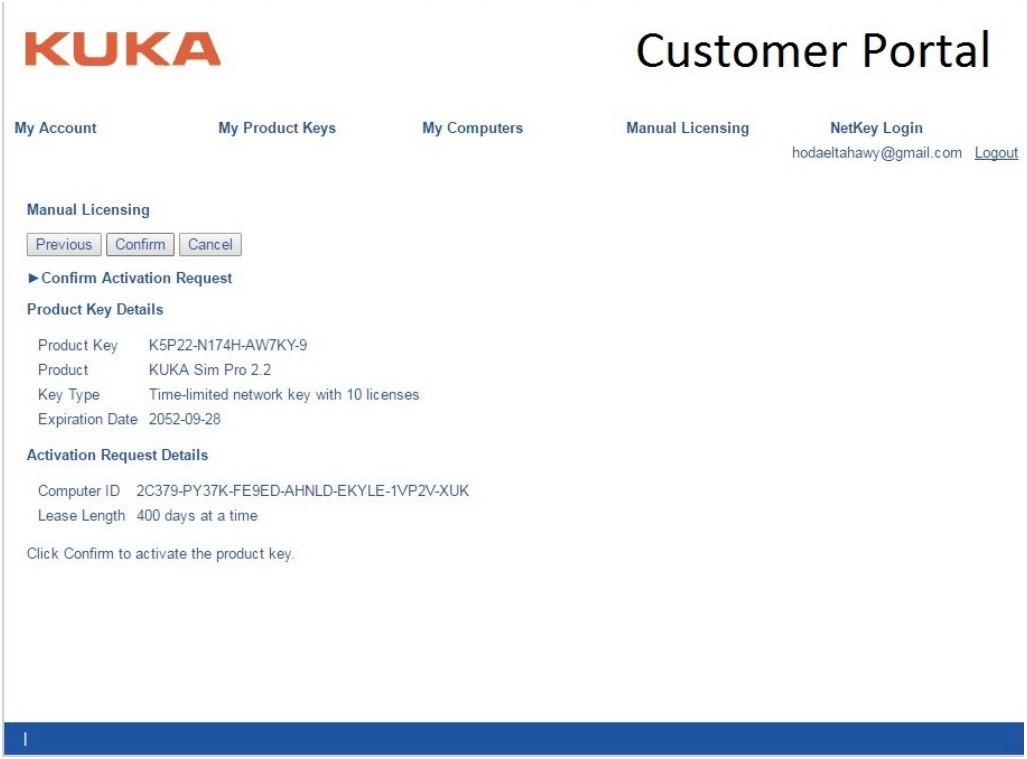
1. Start the installation of KUKA.Sim Pro

2. Enter the license key

3. Select “Activate manually” and save the request file

4. Go to the Visual components customer portal and use the given email and password, mentioned in the previous section, to login.

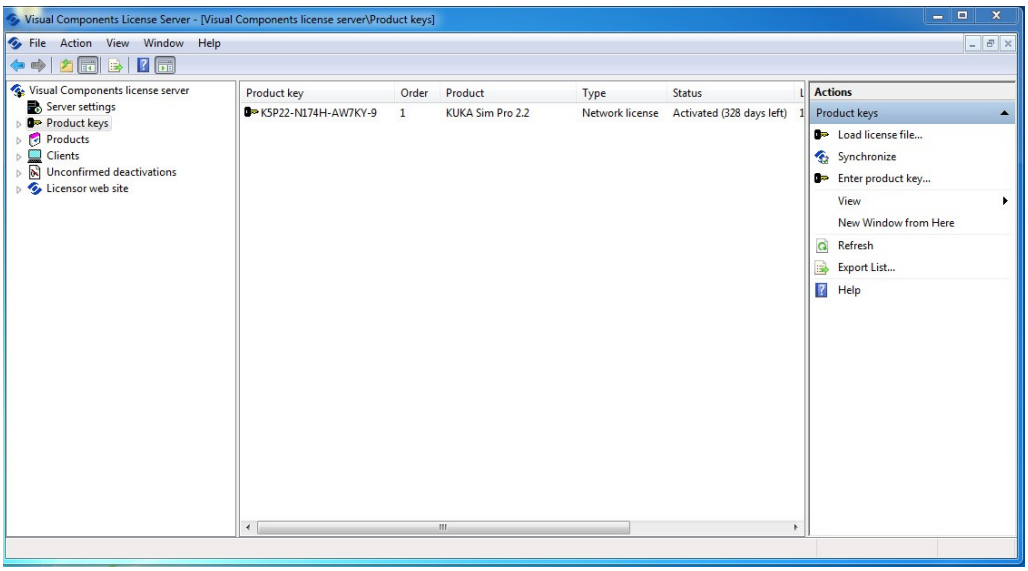
5. On the website, choose Manual Licensing, upload the request file and confirm.



6. The license should be activated.

7. Download the license (.dat) file and click Finish. Please complete the installation steps of the license server before proceeding with the next steps.

8. The license (.dat) file should be loaded into the license server, not the KUKA.Sim Pro interface, in order to complete the activation process.



9. After this process is completed, the network server interface should appear as follows

Installing the license server

Requirements "Microsoft Management Console" (MMC) must be installed on the license server. The software can be downloaded from the Microsoft website. In addition, ".NET framework 3.5" or higher should be installed.

Installation

1. The license server is started via "Start > Programs > Visual Components > Visual Components License Server > Visual Components License Server Manager".
2. Choose "Server settings" from the left panel, make sure that the license server is started, if not, click Start to activate it. The port number value is 5093.
3. On the left-hand side, select "Product keys" in order to enter the license key.
   - The right-hand side changes and "Enter product key..." is offered. Click on this.
   - A window is opened in the center, in which the license key can be entered.
   - If manual licensing is being performed using a license file, this license file must be loaded with "Load license file...".
4. Enter the license key and confirm with OK.

For network licensing, an account linked with the purchase is created on the Visual Components website (http://www.visualcomponents.com/). In the specified customer portal (https://portal.visualcomponents.net/website/Login.aspx), sign in with the

email hodaeltahawy@gmail.com and password quails@123 . In the “My Products keys” tab, you will find the product key for KUKA.Sim Pro on the KUKA-PC device, at the Mechatronics lab. The license is already activated and will only require renewal after a period of 400 days starting 3-12-2016, which is on 7-1-2018.

3.5. End-effectors installation

3.5.1. Pneumatic gripper

The KUKA AGILUS offers numerous options for different end-effectors installations. One of the most common end effectors is a gripper, whether it be vacuum, pneumatic, hydraulic or servo-electric grippers. We are concerned with pneumatic grippers, which operate using air pressure. When air pressure is applied on the pistons, the gripper closes. When the pressure is released the gripper opens. The only way to manage the force in the gripper is to manage the air pressure in the air intake (or valve). The gripper used in our study is the SOMMER automatic GP 404 NC-C We also had the opportunity to work with one of KUKA Roboter's Application software; Gripper&SpotTech. These are ready-made software packages created for different industrial applications. Optional features can be installed on the controller easily and quickly and can also be tailored to the specific production environment.

These software include, and not limited to, KUKA.ArcTech, which enables implementation and programming of applications for arc welding and plasma cutting, KUKA.ConveyorTech, which automatically adapts the actions of the robot to the motion of an assembly line or conveyor belt and KUKA.CNC, which links the CNC and robot directly to each other. As a result, they can be operated like a conventional CNC controller.

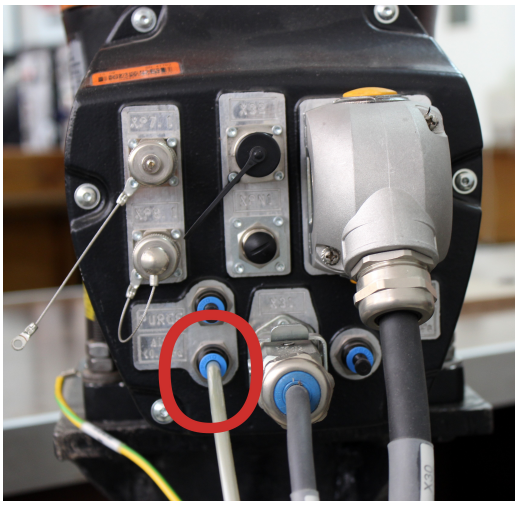
Pneumatic grippers can be used in many applications, some of which include assembly purposes, Labs automation, and on mobile robots.

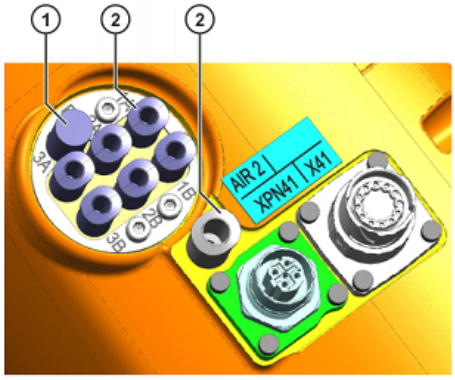
This software package offers numerous advantages, including:

- 16 freely configurable grippers
- 16 freely configurable grippers
- Gripper conditions statically and dynamically monitored
- Unlimited user-defined gripper icons
- Unlimited user-defined gripper icons
- Graphical user interface with indicator lamps, a status display and online adaptation
- Adaptation via WorkVisual 4.0 and on the smartPAD for production-relevant elements

3.5.1.1. Gripper connection

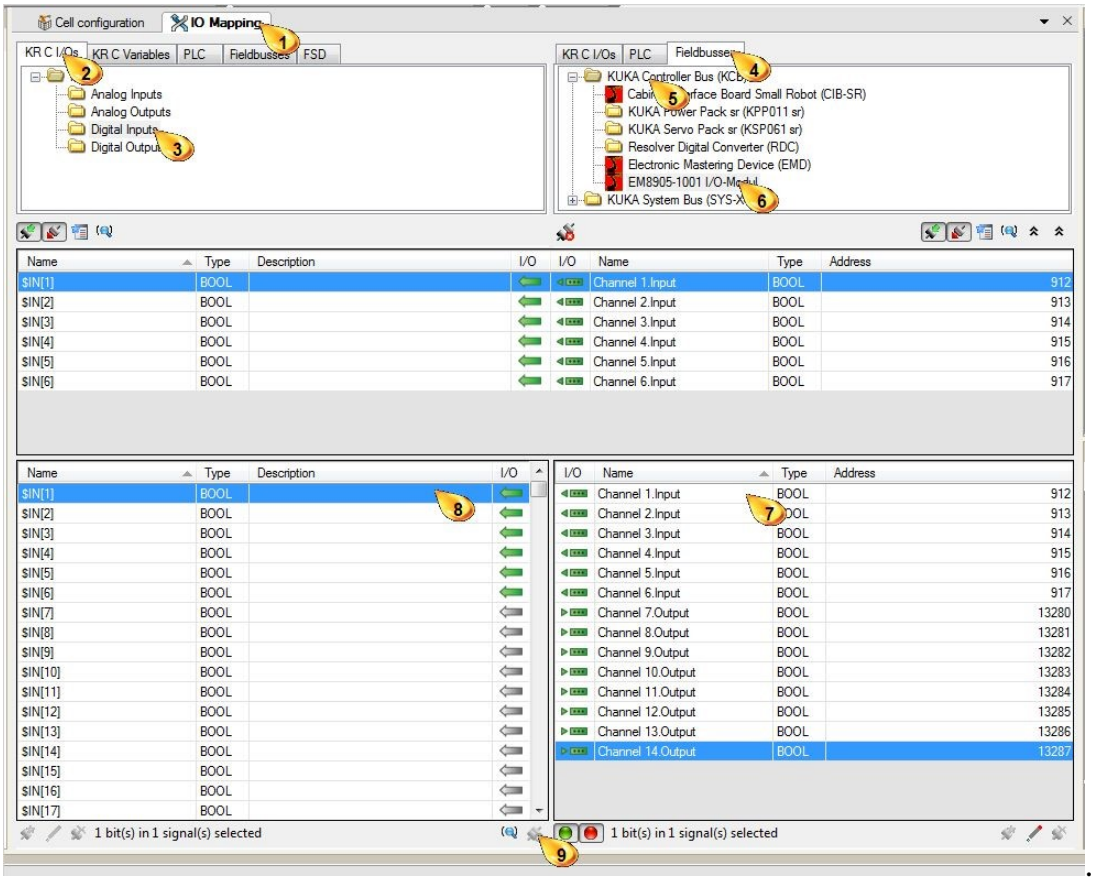
The gripper is connected to the air supply through the Air 1 port located on link 3 (The port is shown in the following picture). This port is connected internally to Air 1 port on the back of the robot arm, which in turn is connected with the air compressor.





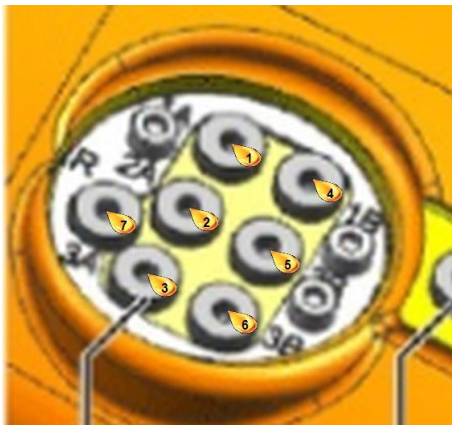
In order for these ports to be operative, the connection between the ports and the controller must be activated, this is performed using mapping. Mapping can be simply described as the process of creating a link or a connection between ports on the robot arm and inner components in the controller in order to use them in control purposes, in our case operating the gripper. Mapping is performed through WorkVisual through the steps mentioned below.

1. get current project from robot using workvisual
2. save it as different file name
3. activate project in work visual (by double clicking on Controller<kss version>)
4. open I/O mapping
5. leave left pane on KRC (those are I/Os that robot programs can access), and click on inputs
6. move right pane to Fieldbus (those are physical I/O), then highlight EM8905 module (this is I/O card inside agilus arm)
7. map inputs of EM8905 to robot inputs of your choice
8. repeat steps 5.6.7 for outputs
9. on the robot login as Expert or higher (Expert will work in this case since we did not modify safety configuration)
10. deploy modified project to robot and activate it (on robot). You should have something like on image below.



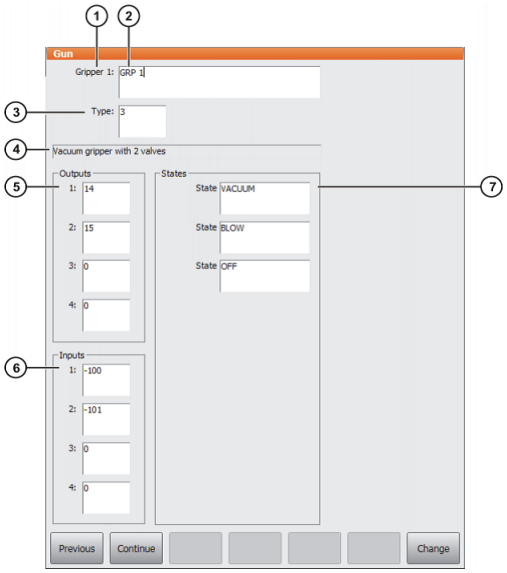
Six valves are mapped to ports:

1. 1A
2. 2A
3. 3A
4. 1B
5. 2B
6. 3B
7. R (relief/exhaust)



Configuring predefined grippers

Gripper settings can be changed using smart pad. In the main menu, select Configure > I/O > Gripper. A window opens (shown in figure), inside it you'll find a list of the predefined grippers, select the desired gripper number with Next or Previous.
You can change number of grippers)1), Name of gripper (2), Type of gripper (3), designation of gripper type (4), Assignment of the output numbers (5), Assignment of the input numbers (6) and Switching states (7).
The third cell, designated for the type of gripper is explained in the following section, Predefined gripper types.

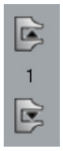


Predefined gripper types There are five predefined gripper types in Gripper&SpotTech. If these types are not sufficient, additional gripper functions can be programmed.

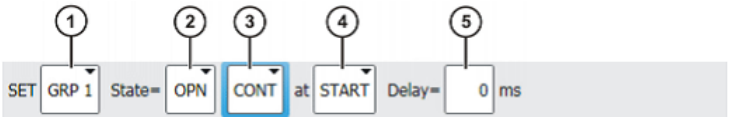
- Type 1: Single-element gripper, static, open/closed
- Type 2: With mid-position valve
- Type 3: Vacuum gripper with 2 valves
- Type 4: Vacuum gripper with 3 valves
- Type 5: Single-element gripper with pulse valves, open/close

More information about the specifications of each type and user specified grippers can be found in manual "KST_GripperSpotTech_32_en".

Gripper operation Manual gripper control can be performed using technology keys on smart pad. The settings for the technology keys are already set for this gripper and appear with the following icons on the smart pad screen, next to the assigned buttons.

Icon	Description
	Select gripper The number of the gripper is displayed. Pressing the upper key counts upwards. Pressing the lower key counts downwards.
	Toggle between the gripper states (e.g. open or close)

The gripper is opened or closed using these buttons, after pressing the enable buttons on the back of the smart pad. They can also controlled inside KRL programs in inline form through the following command



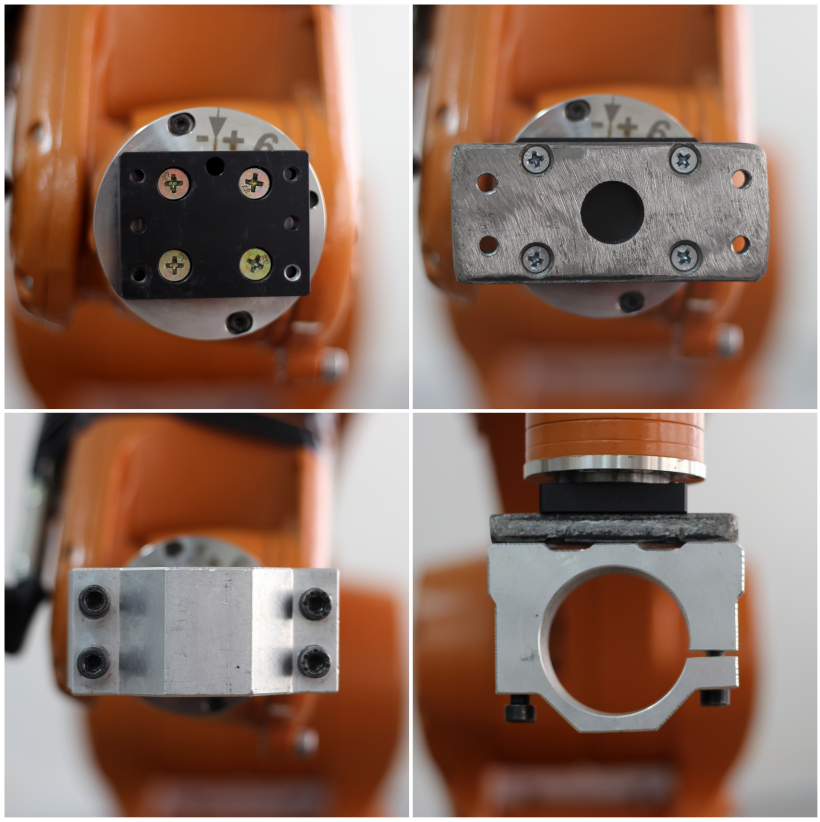
Where

1. Choosing the desired gripper from a list of predefined grippers in settings
2. Set the state of the gripper, whether to open or close
3. CONT: Execution in the advance run
4. Box only available if CONT selected.
   - START: The gripper action is executed at the start point of the motion.
   - END: The gripper action is executed at the end point of the motion.
5. Box only available if CONT selected. Define a wait time (-200:200 ms), relative to the start or end point of the motion, for execution of the gripper action.
6. Box only available if [blank] selected. Data set with gripper parameters

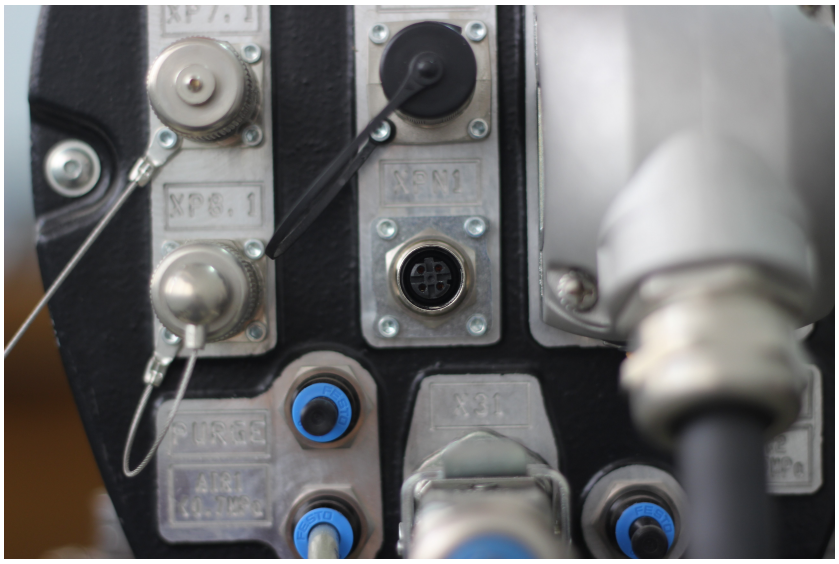
3.5.2. Electric spindle

For the purpose of milling, a spindle was attached as an end effector to perform this process. The spindle used was a simple ON/OFF spindle, which required no control signals, so it merely needed to be attached to the end effector and to an external power supply.

Spindle attachment For the spindle to be attached to the sixth axis of the robot, a metal linkage was designed and manufactured to fit both the spindle and the mounting surface on the sixth axis. The first picture shows the mounting surface of the sixth axis of the robot. The second shows the metal linkage after being installed to the mounting surface. The Third and fourth pictures shows the spindle's metal holder.



Spindle connection The power supply of the spindle is connected to port XPN1 on the fourth axis, whose output is internally linked to a similar port XPN1 on the back side of the robot. The port contains four openings or smaller ports; the positive wire is connected to the two right-hand side ports and the negative side to the left-hand side ports. The ports are shown in the picture below.



Spindle specifications The spindle used is an Air cooled spindle, with a 300W CNC Spindle Motor, supplied by a 220 voltage source, with adjustable speed through the attahced knob. The spindle must operate with full speed during the milling process. Further information about the spindle can be found in the resources in the references.

End mill used for machining The end mill used is a double blade 6 mm Carbide blade. A 6 mm collet was used to attach the end mill to the spindle. A collet is a subtype of chuck that forms a collar around an object to be held and exerts a strong clamping force on the object when it is tightened, usually by means of a tapered outer collar. Both the end mill and the collet can be changed for different milling purposes, as an example, a smaller end mill can be used to obtain a higher level of fine details that larger end mills can't offer, while larger end mills can be used to remove more material or perform faster in basic milling operations that does not require a high level of details.

"It is impossible for us, who live in the latter ages of the world, to make observations in criticism, morality, or in any art or science, which have not been touched upon by others. We have little else left us but to represent the common sense of mankind in more strong, more beautiful, or more uncommon lights."

— Joseph Addison, (English essayist, poet, and politician, 1672–1719), Spectator, No. 253

Chapter 4

Robot Programming

4.1. Cartesian–Axis Specific Coordinate System

4.1.1. Coordinate systems in conjunction with robots

The following Cartesian coordinate systems are defined in the robot controller:

WORLDCoordinate System Fixed, rectangular coordinate system whose origin is located at the base of the robot. It is the root coordinate system for the ROBROOT and BASE coordinate systems. By default, the WORLD coordinate system is located at the robot base.

ROBROOT Coordinate System Fixed, rectangular coordinate system whose origin is located at the base of the robot. It is the root coordinate system for the ROBROOT and BASE coordinate systems. By default, the WORLD coordinate system is located at the robot base.

BASE Coordinate System Fixed, rectangular coordinate system whose origin is located at the base of the robot. It is the root coordinate system for the ROBROOT and BASE coordinate systems. By default, the WORLD coordinate system is located at the robot base.

TOOL Coordinate System a Cartesian coordinate system which is located at the tool center by default, the origin of the TOOL coordinate system is located at the flange center point. The TOOL coordinate system is offset to the tool center point by the user

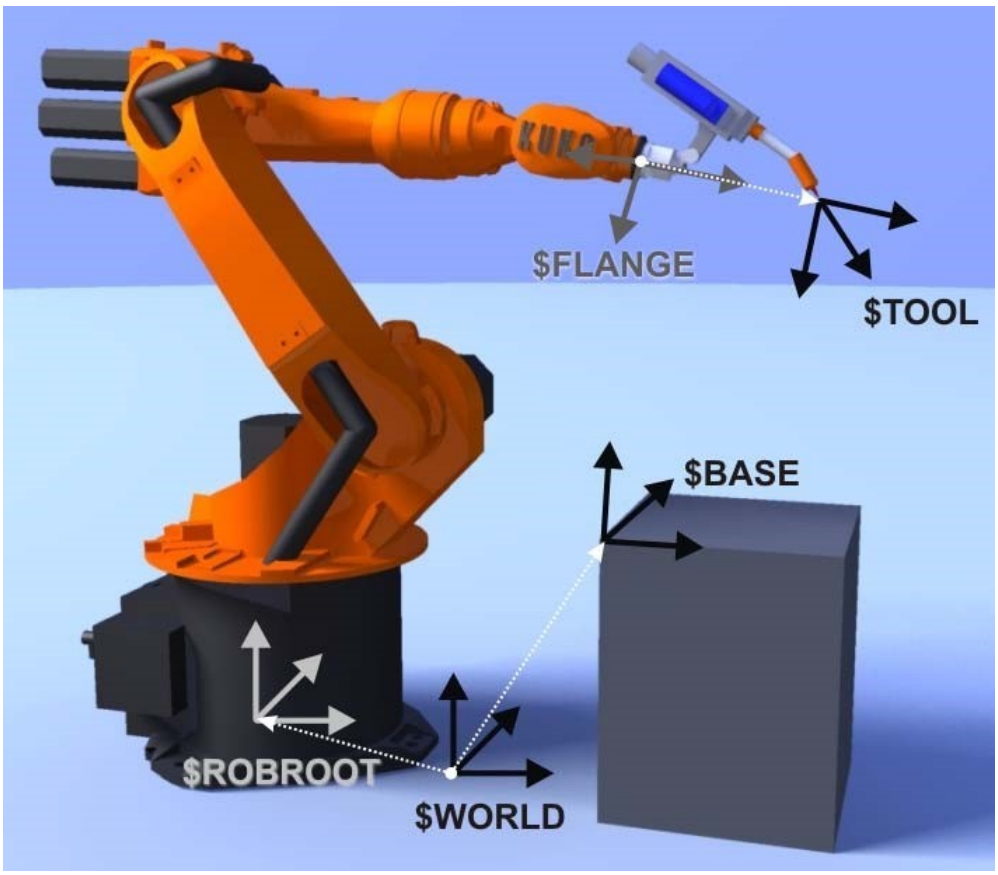


Figure 4.1.: KUKA robot coordinate systems

4.2. KUKA Robot Language (KRL) Quick Guide

KRC 4 controller uses KRL KUKA programming language.

4.2.1. Variables and Declarations

All system variables start with $ sign, mind not starting any "user-defined" name with this sign to avoid syntax errors.

Names in KRL

- Can have a maximum length of 24 characters
- Can consist of letters(A - Z), numbers(0 - 9) and the special characters '$'.
- Must not begin with a number.
- Must not be a keyword.

Declaration and initialization of variables

- Variables (simple and complex) must be declared in the SRC file before the INI line and initialized after the INI line
- Variables can optionally also be declared and initialized in a local or global data list.
- Every variable is linked to specific data type.
- The data type must be declared before use.
- The keyword for the declaration is DECL. It can be omitted in case of the four simple data type
  - In order to place syntax before the INI line, the DEF line must be activated:

Open file >Edit >View >DEF line

```
DEF programName()
    DECL data type user defined variable
    ;declaration section of  variables
    INI
    ;Initialization section of user defined variables.
    ...
    ;Instruction Section
    ...
END
```

Simple Data types

Data Type	Keyword	Meaning	Range	Example
Integer	INT	integer number	$-2^31 \ldots 2^31-1$	2
Real	REAL	floating point number	$\pm 1.1E-38 \cdots \pm 3.4E+38$	4.23
Boolean	BOOL	logic state	TRUE, FALSE	TRUE
Character	CHAR	character	ASCII character	C

Table 4.1.: KRL Data Types

Structure Types

- AXIS: A1 to A6 are angle values (rotational axes) or translation values (translational axes)

```
Axis:  A1 .., A2 .., A3 .., A4, A5 .., A6 ..
```

- FRAME: X, Y, and Z are space coordinates, while A, B, and C are the orientation of the coordinate system.

```
FRAME: X .., Y .., Z .., A .., B .., C ..
```

- POS and E6POS: S (Status) and T (Turn) define axis positions unambiguously

```
POS: X .., Y .., Z .., A .., B .., C .., S ..., T
```

4.3. Motion Programming

4.3.1. Motion Types

The robot can move in various motion types. Paths are created according to the operation of each axis. Thus, the robot can be controlled to create either linear or circular path.

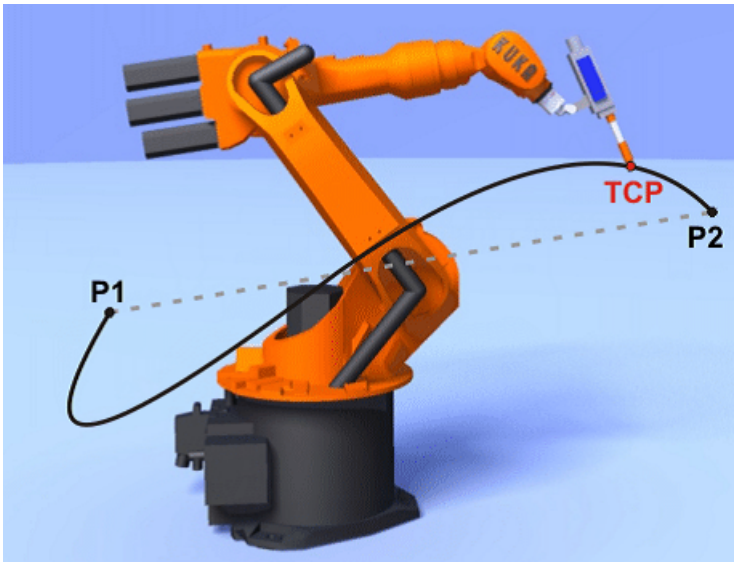
4.3.1.1. Axis-specific motion

The robot guides the TCP along the fastest path to the end point. The fastest path is generally not the shortest path and is thus not a straight line. The first motion in the program must be PTP as status and turns are only evaluated here. The coordinates of the end point are absolute.

characteristics

- smooth motion
- Robot can move from start to end singularity free. As long as both the starting and ending points are in the working envelope, the robot will get to the end point without collision or sudden movement.
- Control is much simpler than continuous path control.

Figure 4.2.: PTP Motion



4.3.2. CP motion

LIN Motion Motion at a defined velocity and acceleration along a straight line. This motion requires the programmer to "teach" one point. The robot uses the point defined in the previous move as the start point and the point defined in the current command as the end point and interpolates a straight line in between the two points.

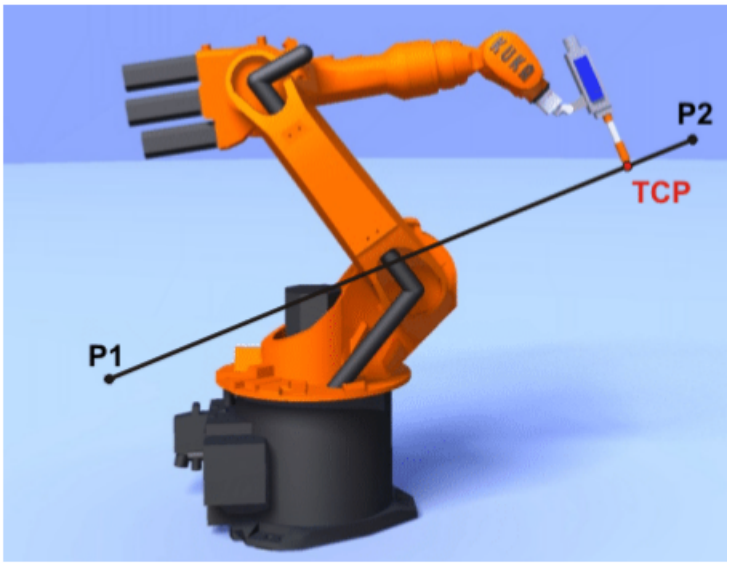


Figure 4.3.: LIN Motion

CIRC Motion Motion at a defined velocity and acceleration along a circular path or a portion of a circular path. This motion requires the programmer to "teach" two points, the mid-point and the end point. Using the start point of the robot (defined as the end point in the previous motion command) the robot interpolates a circular path through the mid-point and to the end point.

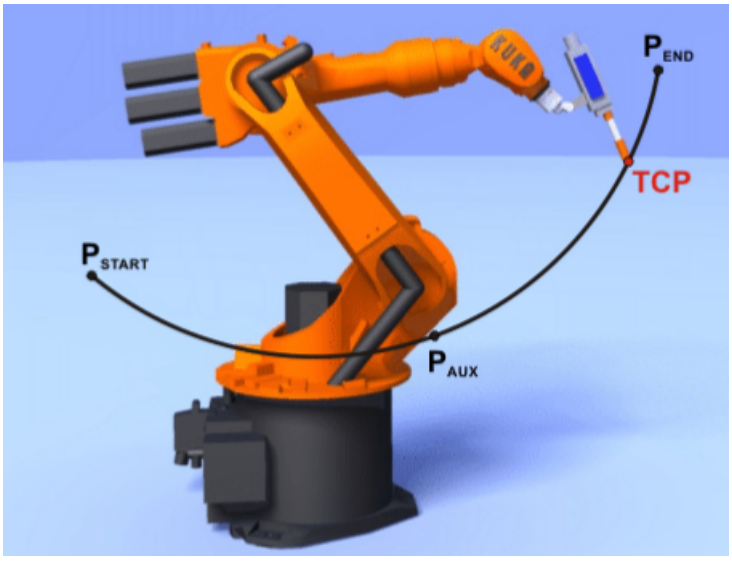


Figure 4.4.: CIRC Motion

4.3.3. Approximate Positioning

Approximate positioning of motion means that the next programmed point will not be exactly reached. This can help to shorten cycle times

4.3.3.1. PTP-PTP approximate positioning

For the purposes of PTP approximate positioning,the controller calculates the distances the axes are to move in the approximate positioning range, and plans velocity profiles for each axis which ensure tangential transition from the individual instructions to the approximate positioning contour.

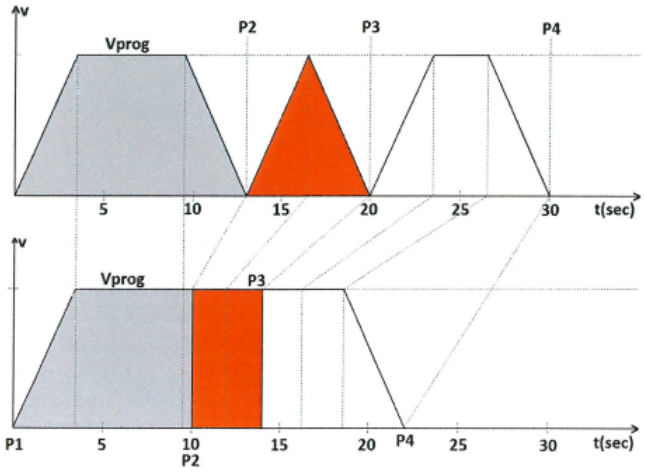


Figure 4.5.: Speed Profile: a) If all points approached exactly and b)In case of approximate positioning of the points

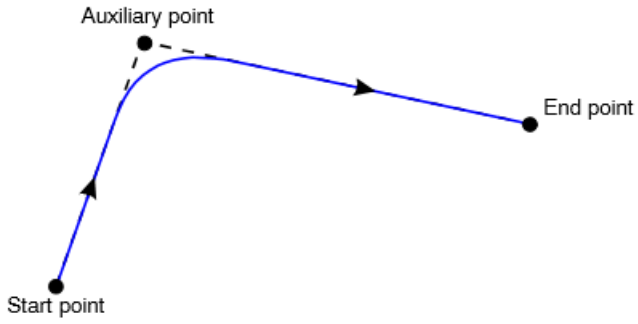


Figure 4.6.: Approximate positioning of an auxiliary points

System Variable, $APO.CPTP enables the start of approximate positioning to be specified as a percentage of these maximum values. The approximate positioning of a point is displayed in the PTP command by adding the key word C_PTP:

```
$ APO.CPTP = 80
PTP HOME C_PTP
```

The greater this value the, the more path is rounded.

Status and Turns The position of x,y,z and orientation A,B,C values of TCP are not sufficient to define the robot position ,as different axis

positioning are possible for the same TCP . Status and turns serve to define the position that can be achieved with different axis positions.

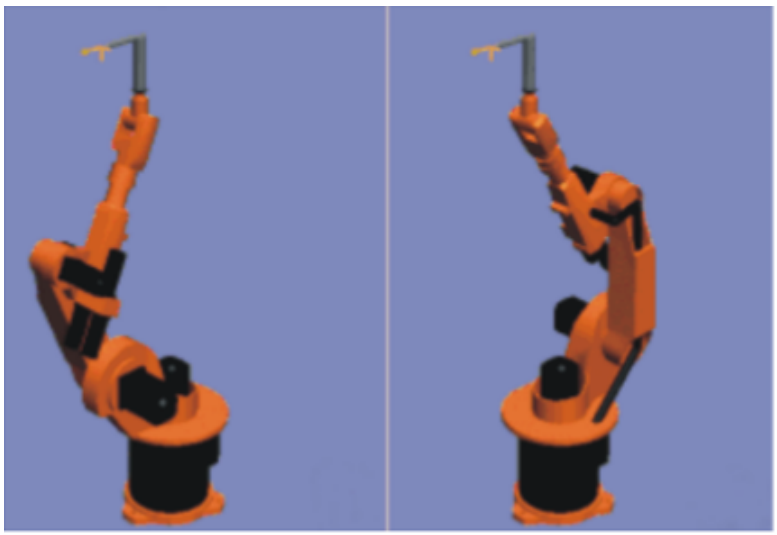


Figure 4.7.: Same TCP, different axis position

4.3.4. User Programming

Inline forms are available in the KSS for frequently used instruction. They simplify programming and facilitates user interface with controller without the need of knowing detail information about KUKA programming Language

4.3.5. Expert Programming

In the Expert interface, can achieve advanced programming using the KRL programming language and perform complex application programs including subprograms, interrupt programming, loops, and program branches.

"Look up at the stars and not down at your feet. Try to make sense of what you see, and wonder about what makes the universe exist. Be curious."

— Stephen Hawking, (British theoretical physicist, and cosmologist

Chapter 5
Robotic Operating System (ROS)

Recently, robotics community witnesses excessive progress. In Spite of this progress, It still undergo complexity and significant challenges to the software developers; One of the main reasons for this issue is handling the wide variation in tasks and environments of robot systems by an individual. ROS -Robot Operating System- is the flexible framework that makes robot software developing is more robust and accessible by robotics community. The official description of ROS is:

> "ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers."

5.1. Modularity

The core of successive software algorithm is the ability to reuse in different projects without reimplementing. Modularity is an important Software principle. It divides complex systems into manageable and

simpler modules ROS adds value to the most robotics projects and applications because of it Modularity, so you can use ROS as much as you desire and still implement your own parts.

5.2. Distributed Nature

Communication between multiple process is the key to powerful system. ROS provides an integration point that able to manage hardware, drives, development tools, useful libraries, simulators and much more. A ROS distribution is a set of ROS software packages that can be downloaded to your computer.

5.3. Road Map to ROS development

5.3.1. Filesystem Level

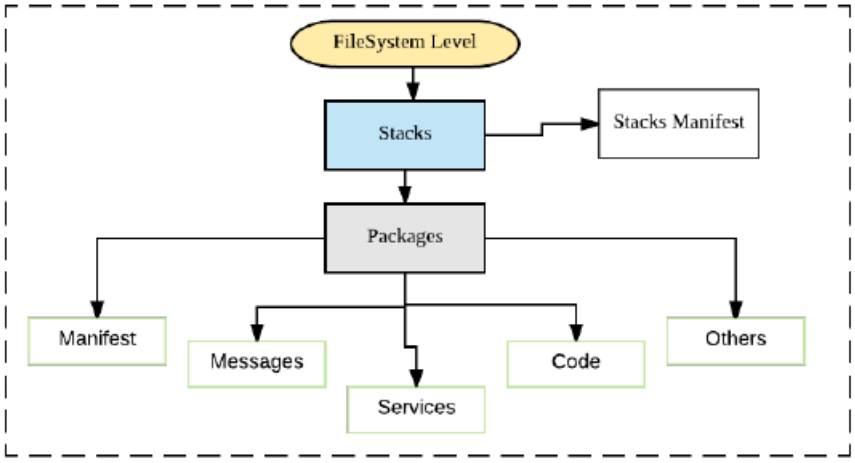
The filesystem level explains how ROS files are organized on the hard disk.ROS program is divided into folders, and these Folders has some files that describe their functionality

label description

Packages A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module.

ROS Package tools To create, modify, or work with packages, ROS gives us some tools for assistance

Figure 5.1.: Filesystem level representation



Command	Function
`rospack`	This command is used to get information or find packages in the system
`roscreate-pkg`	creating new package with your own dependencies
`rosmake`	This command is used to compile package
`rosdep`	This command installs system dependencies of a package
`rxdeps`	This command is used if you want to see the package dependencies as a graph

Table 5.1.: ROS Package commands

Stacks Packages in ROS are organized into ROS stacks.the main goal of stacks is to ease the process of sharing codes.

Messages Nodes communicate with each other by publishing messages to topics. A message is a simple data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.)

Services ROS uses simplified service description language for describing ROS service types. This builds directly upon the ROS msg format to enable request/response communication between nodes. Service descriptions are stored in .srv files.

5.3.2. Computaional Graph Level

The basic Computation Graph concepts of ROS are: nodes, Master, Parameter Server, messages, services, topics, and bags, all of which provide data to the Graph in different ways.

- Nodes: Nodes are processes where computation is done. A system better has many nodes to control different functions. Nodes are written with ROS client library; roscpp, or rospy.
- Master: ROS Master is the part that facilitates all the communication between nodes.You should allow the master to continue running for the entire time that you're using ROS.
- Parameter Server: The Parameter Server gives us the possibility to have data stored using keys in a central location.
- Messages: The Parameter Server gives us the possibility to have data stored using keys in a central location.
- Topics: Messages are organized into topics. Nodes that need to listen to certain messages will **subscribe** to the topics that it is interested in, or if the node wants to share it is information, the node will **publish**to an appropriate topic.

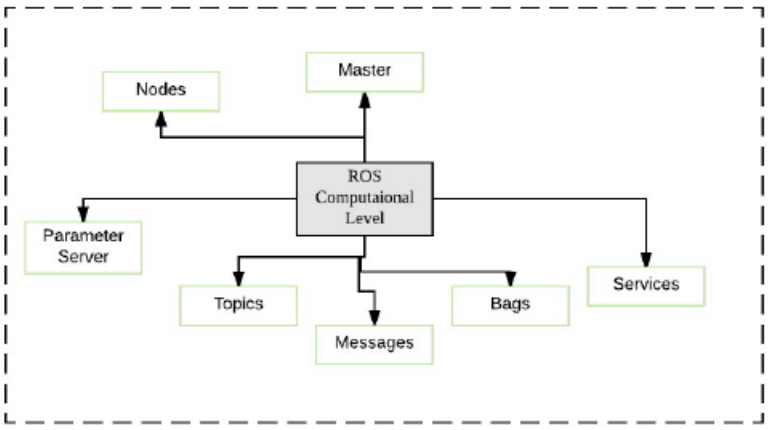


Figure 5.2.: Filesystem level representation

5.3.3. Community Level

The third level is the Community level where you can find useful resources and interact with different developers to exchange knowledge, algorithms, and softwares. The official description of ROS Community level resources:

- *Distributions: ROS Distributions are collections of versioned stacks that you can install. Distributions play a similar role to Linux distributions: they make it easier to install a collection of software, and they also maintain consistent versions across a set of software.*
- *Repositories: ROS relies on a federated network of code repositories, where different institutions can develop and release their own robot software components.*
- *The ROS Wiki: The ROS community Wiki is the main forum for documenting information about ROS. Anyone can sign up for an account and contribute their own documentation, provide corrections or updates, write tutorials, and more.*
- *Bug Ticket System: Please see Tickets for information about file tickets.*
- *Mailing Lists: The ros-users mailing list is the primary communication channel about new updates to ROS, as well as a forum to ask questions about ROS software.*
- *ROS Answers: A Q and A site for answering your ROS-related questions.*
- *Blog: The Willow Garage Blog provides regular updates, including photos and videos.*

ROS tutorials `wiki.ros.org/ROS/Tutorials`

Figure 5.3.: Kinect Xbox 360 with RGB Camera, Depth Camera, and Microphone array

5.4. Using Sensors with ROS: Kinect

5.4.1. Operation and Inferring body position

The process of inferring body position done by two main stages: analyzing a speckle pattern of infrared laser light to produce depth map; then transforms depth image to body part image from motion capture system and finally transforms the body part image into a skeleton.

Figure 5.4.: Speckle Pattern

5.4.2. OpenNI: Natural Interaction

The term natural interaction refers to people communicating with devices through their human senses, such as audio, and vision. The most obvious applications uses this technology are: face/ voice recognition; body motion tracking; and control room electronics using hand gesture

Abstract Layered View The three layered view of OpenNI :

- **Top**: Represents the software that implements natural interaction applications on top of OpenNI.
- **Middle**: Represents OpenNI, providing communication interfaces that interact with both the sensors and the middleware components, that analyze the data from the sensor..
- **Bottom**: Shows the hardware devices that capture the visual and audio elements of the scene.

5.4.3. Skeleton Tracking: User segmentation

Skeleton tracker generate data of the users who exist in the scene; these data includes location of the skeletal joints, and ability to track skeleton position.

There are some consideration for accurate skeleton tracking:

- User considered to be lost if the user is not visible in the scene within 10 seconds
- Users get ID “user 1,2,...”. However the ID is recycled.
- Ideal distance for tracking around 2.5 m
- User should not wear very loose clothing, for better result.

Calibration To start tracking body, the user should do calibration position “psi pose” as shown in figure.

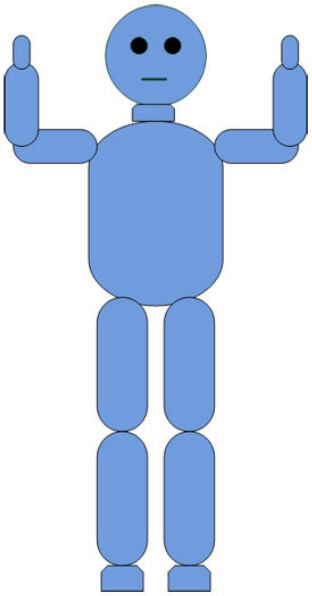


Figure 5.5.: PSI pose

Limitations:

- User should not be sitting
- Most of the user body should be invisible
- User should be located 1 m away from kinect

Skeleton tracker output Skeleton tracker return the position, and quaternion of the joint.

Joint Transformation Joint positions and orientations are given in the real world coordinate system. The origin of the system is at the sensor. The positive direction of X-axis is to the right of origin “, The positive direction of Y-axis is up, and The positive direction of Z-axis is in the direction of increasing depth. The coordinate frame is shown in the figure above.

Keep in mind that representation of the skeleton in Rviz supposed to be in Mirror mode which indicates that the LEFT side labeled in Rviz

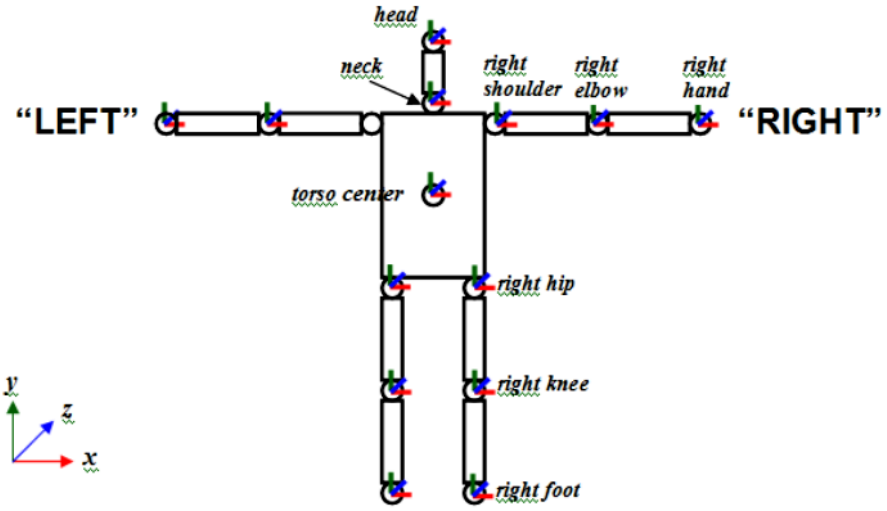


Figure 5.6.: Skeleton joints Coordinate

window is actually your RIGHT side. Joint positions are measured in units of m and orientations are in radians.

5.4.4. Kinect Driver

Openni launch Package This package contains launch files for using OpenNI-compliant devices such as the Microsoft Kinect in ROS. from the device driver into point clouds, disparity images, and other products suitable for processing and visualization.

to open Kinect device and transform raw data to convenient data, run this command:

```
$ roslaunch openni_launch openni.lanuch
```

Openni tracker Package openni_tracker broadcasts the OpenNI skeleton frames using tf. this package allows you to track a person's skeleton using a Kinect. It also gives you the positions, relative to the camera frame run this command:

```
$ rosrun openni_tracker penni_tracker
```

Stand in front of the Kinect. If the terminal window where you ran openni_tracker, you should see output like this:

```
[INFO]: New User 1
[INFO]: Calibration started for user 1

[INFO]: Calibration complete, start tracking user
```

5.4.5. 3D Visualizing

As discussed in the previous sections, Kinect produced 3D data in form of point clouds.For this reason ROS has invented tool to visualize this type of data. these tools enable you to develop robotic system faster by visualizing your data and evaluate the quality of result for validation.

Visualizing 3D data using rviz With roscore running, we only have to execute the following code to start rviz:

```
$ rosrun rviz rviz
```

This command will open rviz interfac

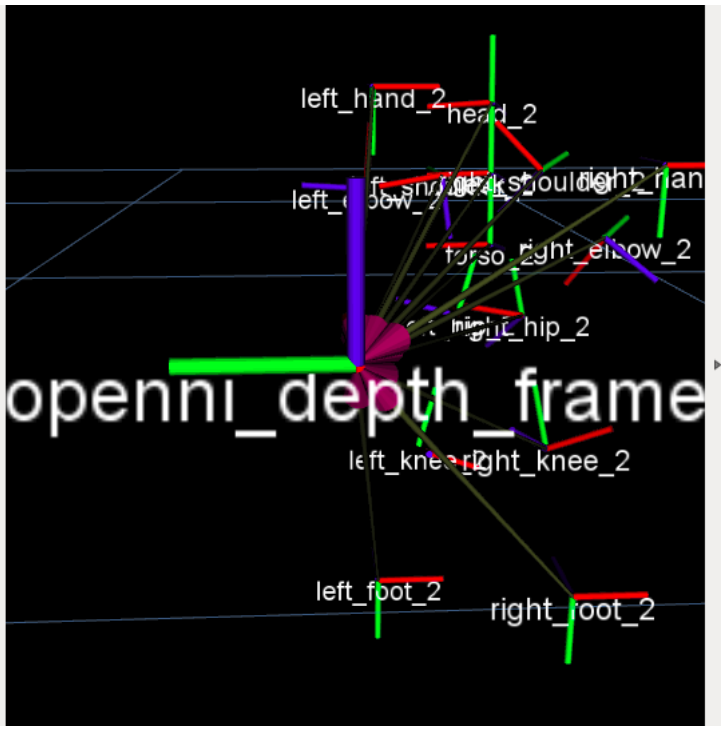


Figure 5.7.: Visualizing skeleton joints using Rviz

"Be sure you put your feet in the right place, then stand firm."

— Abraham Lincoln, (American 16th President, 1809–1865)

Chapter 6
Development

6.1. Development of Industrial Applications: Robot Machining

6.1.1. Introduction

CNC Machining is a process used in the manufacturing sector that involves the use of computers to control machine tools. Tools that can be controlled in this manner include lathes, mills, routers and grinders using CNC machining language (called G-code) that essentially controls all features like feed rate, coordination, location and speeds. There are many advantages to using CNC Machining. The process is more precise than manual machining, and can be repeated in exactly the same manner over and over again.

CNC Machine A Conventional CNC Machine is usually designed to do only one type of manufacturing processes. For example, a 3-axis CNC Milling machine is designed to hold a spinning, multi-tooth cutter (spindle) which moves along the three cartesian coordinated to remove material to form a specific structure. This machine is mechanically built to fit the relatively slow and forceful motions required to mill through

hard materials, and converting this machine to do other processes like 3d printing or laser cutting will not give the optimum output.

CNC Robotic Arm Recently robots can perform the exact cuts and movements needed to produce the highest quality of milling process.Milling with a robotic arm is extremely economical; robots can be reassigned to perform other assignments in a shop - arc welding, material handling, etc. In addition, a robotic arm can handle more of the milling task without the need of human intervention. Moreover, the typical 6-axis articulated robot offers more movement flexibility than a normal milling machine; A robot can mill a complex part from multiple angles. Instead of having to reposition and re-clamp the prototype or mold, repeatedly, the robot can remain stationary.

6.1.2. State of the Art

Many robot manufacturers have made it possible for their robots to understand G-Code lines generated from conventional CAM software. KUKA has its own package (KUKA.CNC) which you can buy for about 250 euros and install on your robot controller. If you find this package expensive, you can use CAM software that is able to export toolpaths into KRL directly like KUKA|prc, RoboDK, PowerMill, and SprutCAM.

6.1.3. Implementations

We found it difficult to buy the KUKA.CNC package, and the other KRL exporting programs are not free and also required large amount of RAM to work generate the toolpath and check for singularities. We decided to make a postprocessor software to convert G-Code into KRL without needing any other software or packages except for the conventional CAM software, so we came up with two solutions:

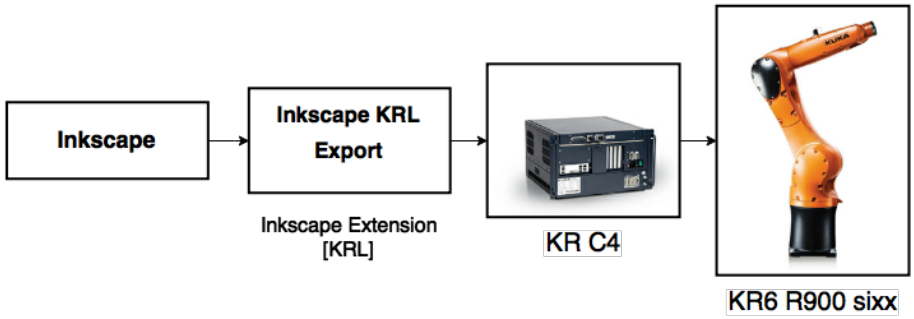
- 2D Solution: “Inkscape KRL Export” extension for converting vectors to KRL.
  Suitable for Writing, Drawing, Laser Cutting or engraving.

- 3D Solution: “ZUKA CNC” a postprocessor tool for converting G-Codes (exported from MeshCAM and ArtCAM) to KRL
  Suitable for Milling, 3D Printing, and Hot Wire Cutting.

6.1.3.1. 2D Machining: KUKA Inkscape Extension

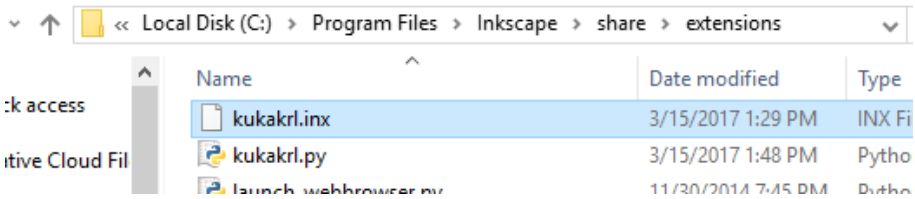
An Inkscape extension, used to convert the contours of text and images in Inkscape to KRL Code to be used with KUKA KR C4 controllers. This work is based on the extension "Repetier G-Code Plugin for Inskscape" which is an open-source G-Code generator (written in Python) that takes vectors as inputs, and generates G-Code lines ready to use at conventional CNC machine. We’ve edited the extension to export KRL instead of G-Code, so that the exported file is ready to be copied and used on the KR C4 control unit. inkscape (2).png

Figure 6.1.: KUKA Inkscape Extension



Adding the extension to Inkscape First you need to install the Inkscape software, which is free to download and use on many platforms. To install the extension:

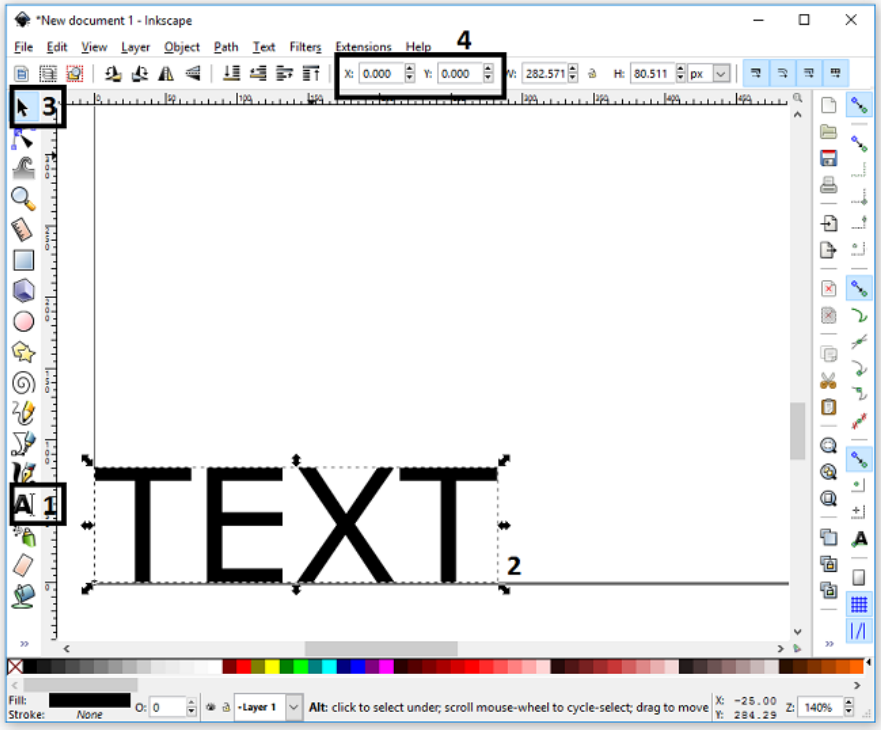
- Go to `https://github.com/mnourgwad/zuka/tree/master/codes/inkscape-kuka`
- Download kukakrl.inx and kukakrl.py
- Copy the files into the extensions directory shown in the picture below.



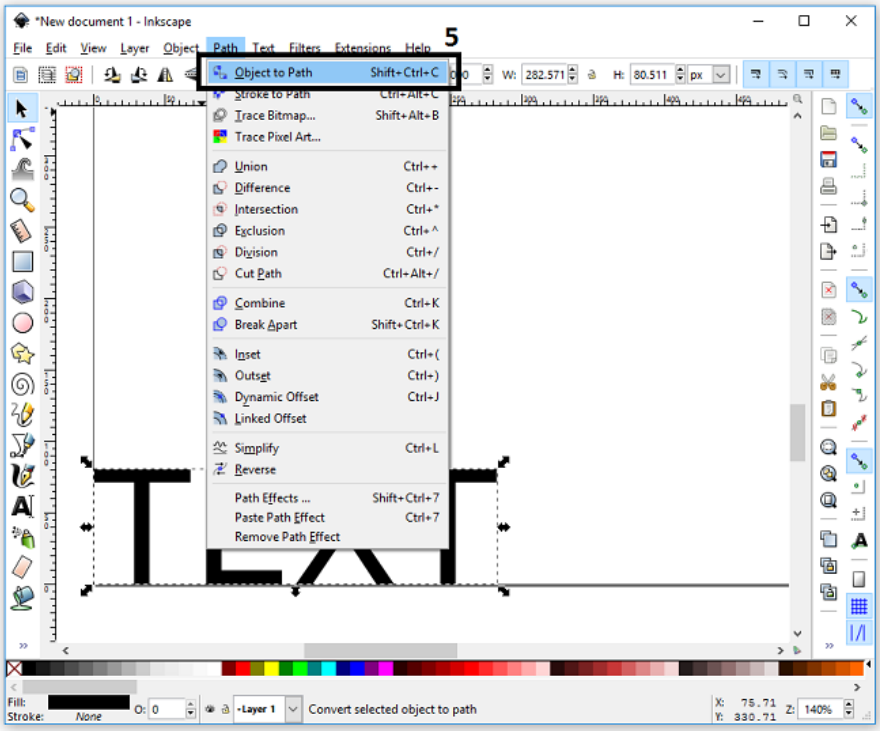
- After a restart of Inkscape, the new extension will be available. You can load it from Extensions >KUKA Tools >Path to KRL

Usage Example: Converting Text to KRL

1. Write your text with the text tool. The bottom left corner is the 0,0 location of the defined base or offset.
2. Mark and position your text. If you have more objects (lines, circles, ...) to embed in your KRL Code, you have to mark them all. Only marked objects will be used to generate the KRL Code.



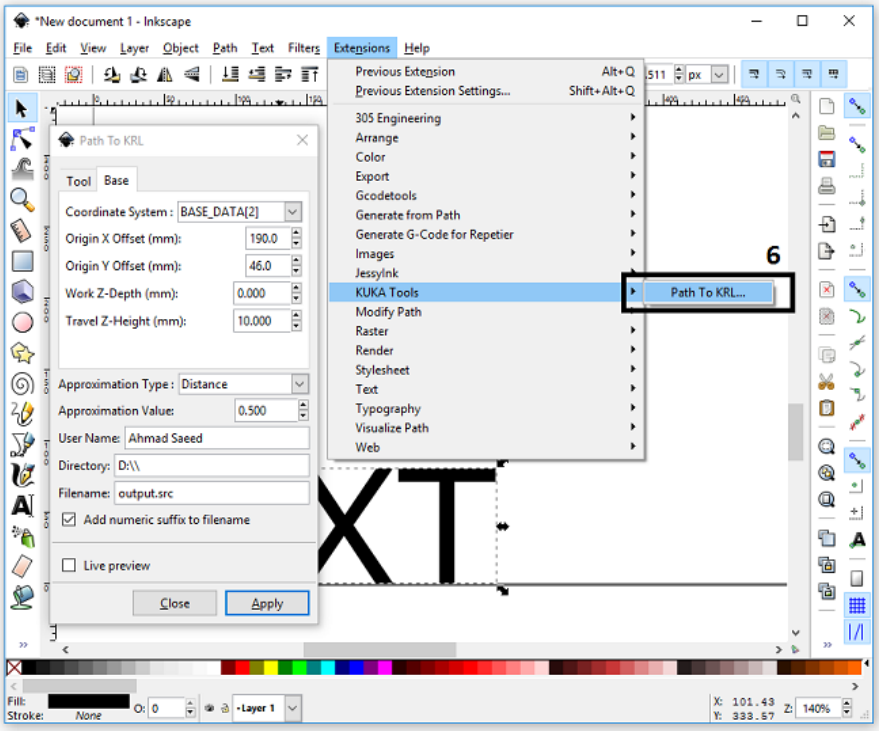
3. Click Path >Object to Path or press Shift + Ctrl + C to convert the text into a path. The Plugin will use this path to generate the KRL Code.



4. Click Extensions >KUKA Tools >Path to KRL to start our plugin.

5. Enter your settings.

6. Click Apply to generate the KRL Code..



7. After that the KRL Code will be stored and the motion path will be outlined.

Dealing with extension settings

Tool Under most circumstances, a KUKA user first defines the TOOL and the BASE then refers to them in KRL using $TOOL, and BASE system variables. The Tool can be defined through one of these two main procedures:

- Automatically be calling the data variables created after performing any of the tool calibration processes from Start-up >Calibrate >Tool. Example: $TOOL = TOOL_DATA[1] where 1 is the saved tool number.
- Manually by entering their numeric XYZABC offset and orientation values. Example: $TOOL = X 280, Y 0, Z -10, A 30, B 90, C 0 where XYZ is the translational offset, and the ABC are Euler angles between the new base and the FLANGE coordinate system. For

this extension, you can only choose the tool number that matches yours, but if you desire to write the values manually, you can edit the created SRC file and add your values.

A° B° C° Orientation Angles (Euler Angles): Defines the relative orientation of the selected tool relative to the selected base in certain motion path. Example: LIN X 0, Y 0, Z 0, A 90, B 180, C 0 This transformation can be performed by:

- Rotating the tool about its z-axis with an angle A (90 degrees)
- Rotating the tool about its y-axis with an angle B (180 degrees)

Note that the ABC angles are the same as Euler angles, but with a little bit different names, as the rotation about tool's z-axis is named C in Euler's.

Work Speed (Feedrate) Defines the velocity at which the robot's TCP is moving while moving according to the desired path. It is expressed in units of Meter Per Second.

Travel Speed Defines the velocity at which the robot's TCP is moving while jumping from a path to another. It is also expressed in units of Meter Per Second.

Coordinate System Defines the desired Base number. The idea behind this is typically similar to the Tool

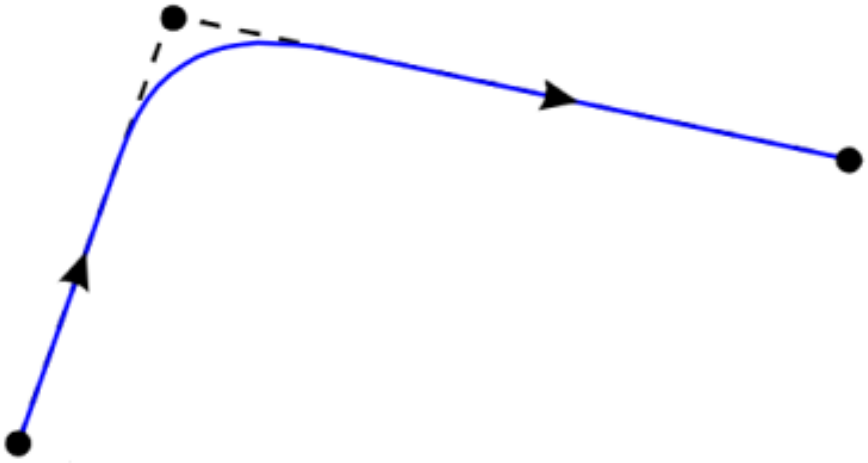
Origin X, and Y Offset Defines the shift from the selected base origin along the x-axis and y-axis. It is expressed in millimeters.

Work Z-Depth Defines the desired TCP's Z-axis value when the TCP is moving according to the desired path. It is expressed in millimeters.

Travel Z-Height Defines the desired TCP's Z-axis value when the TCP is jumping from a path to another. It is expressed in millimeters.

Approximation Type In order to increase velocity, avoid jerky motion, and achieve continuous motion along complex paths, points for which exact positioning is not necessary can be approximated. The robot takes a shortcut as illustrated

Figure 6.2.: Approximation motion



The various approximation motions are:

1. Distance: A translational distance can be assigned to the variable $APO.CDIS. If the approximate positioning is triggered by this variable, the controller leaves the individual block contour, at the earliest, when the distance from the end point falls below the value in $APO.CDIS. Its value is expressed in millimeters.
2. Velocity: A percentage value can be assigned to the variable $APO.CVEL. This value specifies the percentage of the programmed velocity $VEL at which the approximate positioning process is started, at the earliest, in the deceleration phase of the individual block. The component which, during the motion, reaches or comes closest to the programmed velocity value, is then evaluated in terms of translation, swivel and rotation. Its value is expressed in integer number percentage.

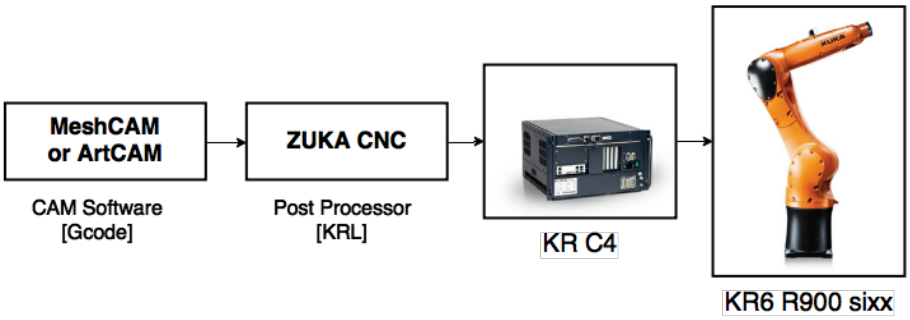
3. Orientation: An orientation distance can be assigned to the variable $APO.CORI. In this case, the individual block contour is left, at the earliest, when the dominant orientation angle (swiveling or rotation of the longitudinal tool axis) falls below the angle distance, defined in $APO.CORI, from the programmed approximate positioning point. Its value is expressed in degrees

Approximation Value:
Defines the value of the selected approximation type. The greater the value of approximation, the more the path is rounded. You can get fine results by setting the its value to 0.5 for Distance type approximation **User Name**
Defines the name of the user making this path. It'll be added as a comment in the code's header. **Directory**
Defines the folder at which the output file will be located **File Name**
Defines the name of the output file. It should be ended with .src **Add numeric suffix to filename**
Adds a number at the end of the file name to allow multiple files with the same name to be saved in the same directory. So for instance, if the file name is output.src, it will be saved like output_0001.src

6.1.3.2. 3D Machining: ZUKA CNC

We noticed that G-Code and KRL are very similar as they consider the workspace as x y z cartesian grid, but they are different in syntax. We made "ZUKA CNC" post processing software, so that the CNC and the robot are linked to each other directly and can thus be operated like a conventional CNC controller. This tool converts conventional G-Code files into a KRL files- ready to be run on KR C4 controller. For CAM processing we used Mesh CAM as it requires no previous knowledge about machining concepts. The worst part of any new CNC software is being confronted by a wall of settings to create a toolpath. MeshCAM has an Automatic Toolpath Wizard that picks as many of those values as possible so that you don't have to. You just pick the cutters, tell MeshCAM the desired quality level, and it will analyze the model to pick values to get you started. If you already know what you're doing you

Figure 6.3.: ZUKA CNC blockdiagram



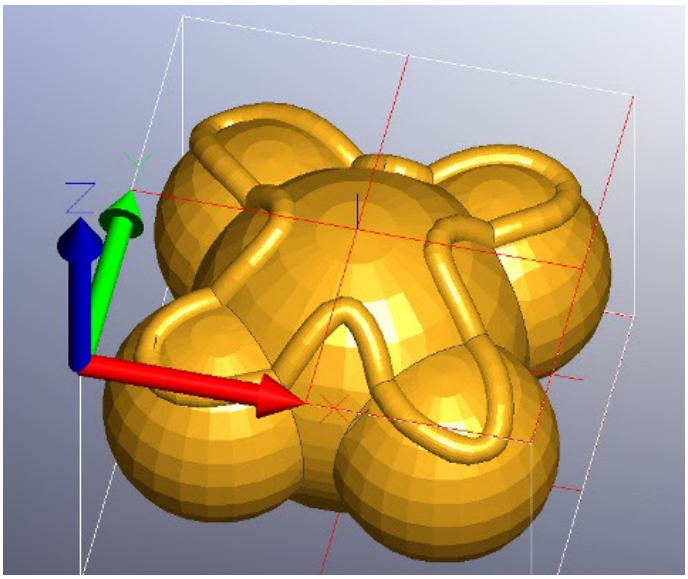
still have complete control over all of your toolpath settings.

Using MeshCAM for generating G-Code (version 6)

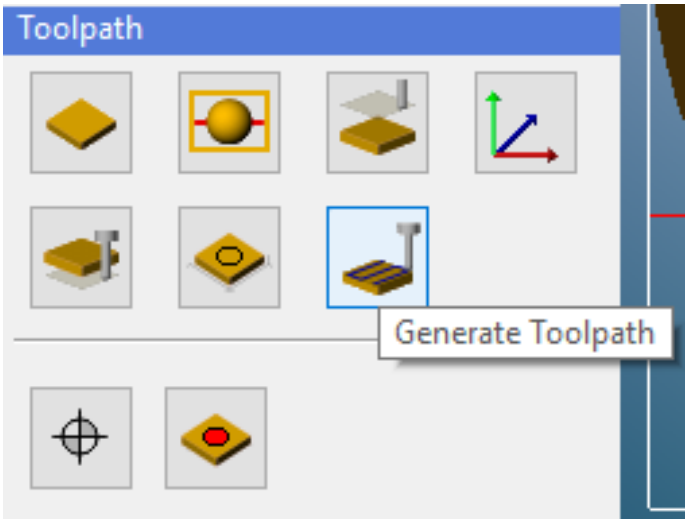
- From your CAD program, export the required file in STL format
- From MeshCAM, import the STL file from “File >Open”
- From MeshCAM, import the STL file from “File >Open”
- Choose “MM” in the dimensions window, if your file has been drawn in mm
- Choose “3 Axis” from the following window
- Now, your model will be loaded into MeshCAM, and the left tool box options will be ready to use
- Use the Geometry tools for moving, scaling, and rotating your model. This is very useful as you can almost fit any model to material size by rotating and scaling.

- - Make sure that the origin (the intersection of x, y, and z) of the model is the same origin of your material stock. For this model, the

origin in on the bottom left side, so the robot origin should be on the bottom left side of the material stock



- If you're sure that all the dimensions are correct, then you are ready to Generate Toolpath.



Before diving into details, you should be aware of some concepts:

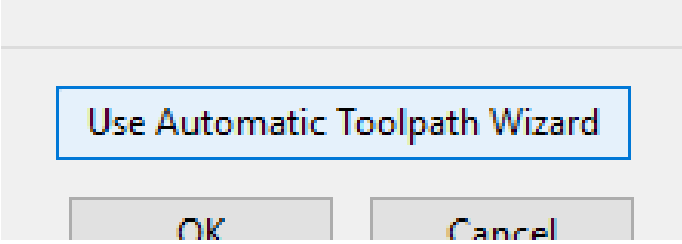
- Roughing
  It's a process of quickly removing large amounts of material. The roughing toolpath divides the geometry up into several layers to

be cut in a sequence. The result will not be satisfying as it will be rough and not containing the fine details of your model

- Finishing
  It's the process of giving the model it's fine details and finish by moving the tool tip in more accurate and slow movements. This process is done after the roughing process
- Depth per Pass
  Defines the depth of each cutting pass for the roughing toolpath. Your "depth per pass" settings can also cause bits to break. As a rule, your depth per pass should never exceed half of your bit's cutting diameter.
- Stepover
  Defines the distance between each cutting line at a given z level. A stepover of between 10% and 20% is typically used for finish machining toolpaths and should give a good surface finish for most materials.
- Feedrate
  Speed of cutting pass. This option will be replaced in the postprocessor with an option called "Working Speed"
- Plunge Rate Speed when plunging the tool into the stock. This option will be replaced in the postprocessor with an option called "Travel Speed".

  Here is a link of the full MeshCAM options
  `http://www.grzsoftware.com/manual/toolpath_settings.htm`

  You can use the Automatic Toolpath Wizard to automatically generate these options



- In order to use the Automatic Toolpath Wizard, you need only to know the dimensions of your End Mill, specially the flute length and diameter

- From the Automatic Toolpath Wizard create a new tool and enter the dimensions of the one you'll use, select the tolerance and quality, then let the Wizard decide the proper parameters.

- Double check for the end mill diameter, stepover, and depth per pass

- Usually you don't need the Waterline and Pencil cleanup Finish processes, so you can uncheck them.

- Click OK and the program will hang for a minute- generating the toolpath

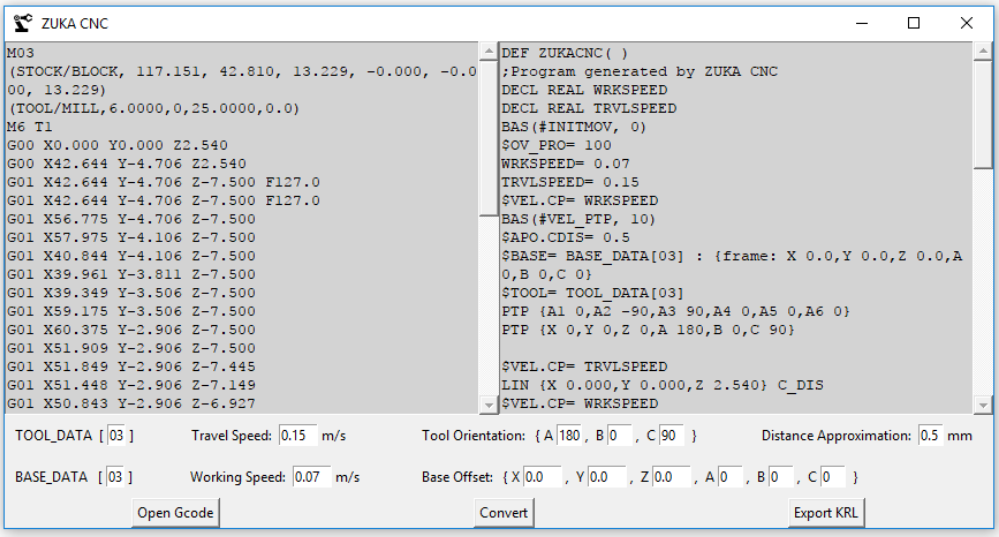
- After it finishes, click Save Toolpath As, and select "TurboCNC V3-MM" from the drop down menu, and by this point you have your G-Code file.

- Open ZUKA CNC and load the G-Code file, convert it, and export the KRL file that you can run on your KR C4 controller

- Please refer to the past section of the Inkscape KUKA Exporter for further details about the parameters used in this postprocessor

- The exported program is ready to run on AUT mode. If you want to run the program on T1 mode, you should change the BAS(#VEL_PTP,10) to BAS(#VEL_PTP,100)

- The operator should run a test of the program to ensure there are no problems. This trial run is referred to as "cutting air" and it is an important step because any mistake with speed, tool position, singularities, or collision could result in a scraped part.



6.2. Development of KUKA communication Interface: KRL Driver

6.2.1. Introduction

Industries that employ robots in a wide variety of applications are the main customers for robot manufacturers. The manipulator market for research applications, on the other hand, is simply too small for the robot manufacturing industry to develop models specifically for such use. While the hardware and mechanical requirements of developed robots are often similar for both industry and research, scientific software requirements are quite different and even contradictory in many aspects. The goal of scientist is to try to gain as much control over the robot as possible, whereas industries seek safe and easy operational interfaces. In particular, although software interfaces that are appropriate for industrial use are available, it is difficult to find interfaces that are applicable for research purposes. The disclosure of the internal control architecture is also very hard to come by. Many manufacturers are unwilling to publish internal details regarding system architecture due to the high levels of competition in the robot market. Consequently, it is not possible to fully exploit many robotic platforms in a scientific context.

6.2.2. State of the Art

Focusing exclusively on Kuka industrial robots, the Kuka Robot Language (KRL) is the standard programming language. It is a text based language that offers data type declaration, specification of simple motions, and interaction with tools and sensors via Input/Output (I/O) operations. It is only possible to run KRL programs on the Kuka Robot Controller (KRC), where program execution is done in accordance with real-time constraints. While the KRL offers an interface that is easy to use in industrial applications, it is quite limited for research purposes. In particular, the KRL is tailored to the underlying controller and consequently, only a fixed, controller-specific set of instructions is offered. Advanced mathematical tools such as matrix operations, optimization or filtering methods are not supported, thus making the implementation of novel control approaches very difficult. There is no native way to include third party libraries and as such, extending the KRL to include new instructions and functionalities is an arduous task. Moreover, it is not possible to directly use external input devices. The standard workaround for partially expanding the robot's capabilities is to use supplementary software packages provided by Kuka. Some examples of such packages are the Kuka.RobotSensorInterface , which allows the manipulator motion or program execution to be influenced by sensor data, and the Kuka.Ethernet KRL XML, a module that allows the connection of the robot controller with up to nine external systems (e.g. sensors). However, several drawbacks accompany these supplementary software packages: I/O is limited, a narrow set of functions is present and major capital investments are often required to actually purchase these packages from Kuka.

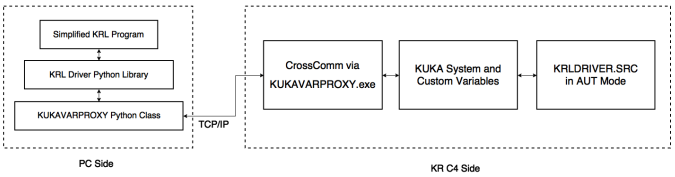
To overcome these problems, an Italian company called "IMTS S.r.L." has hacked into the KUKA KR C controller and made it possible for any device to connect through a TCP/IP protocol to read and write the values of the Global system variables. Since then, many researchers have developed interfaces for this connection like OpenShowVar and JOpenShowVar, but these tools are mainly used to track the robot variables during operation, and move it in limited ways.

6.2.3. Implementation

We started developing an open-source PC interface for KUKA robots programming. This interface makes it possible to program the robot through Python -which is a really good language in dealing with different data types, matrices, and complex calculations, and thus many applications like Artificial Intelligence, Computer Vision, and Machine Learning can be easily implemented. This interface is making use of the KUKAPROXYVAR proxy server, which is an TCP/IP interface to Kuka robots that allows for reading and writing variables and data structures of the controlled manipulators.

6.2.3.1. Operation

Figure 6.4.: KRL Driver System



The kukavarproxy is a TCP/IP server that listens for network messages on the TCP port 7000, the reads and writes data to the KRC system variables. We installed the KUKAVARPROXY, created custom system variables on the KUKA side, and wrote python class and library- on the PC side- to communicate with KUKAVARPROXY to read or change the value of these variables. We also wrote a KRL program- on the KUKA side- to read these variables and act according to them. This made it possible for controlling the robot through normal python code, and opened new world of easy and flexible possibilities for users

6.2.3.2. KRL Driver

Usage The KRC robot controller runs the Microsoft Windows operating system. The teach pendant shows an “HMI” which is a program that

KUKA developed to run on Windows and it is the interface that the robot user has to manipulate the robot through. In order to establish an Ethernet (TCP/IP) connection, you first need to run kukavarproxy on the controllers operating system, then configure the network connection from KUKA "HMI".

Note

You can get the libraries from:
`https://github.com/mnourgwad/zuka/tree/master/codes/kukavarproxy-msg-format`

Copying kukavarproxy to the operating system on the KRC:

- Get the kukavarproxy from *any* of these two sites:
  - `https://sourceforge.net/projects/openshowvar/files/openshowvar/REV%200.13.0/kukavarproxy-6_1.rar/download`
  - `https://github.com/aauc-mechlab/JOpenShowVar/tree/master/KUKAVARPROXY%20rev%206.1.0.101`
- Unpack and copy the folder to a USB flash drive
- Plug it to the KRC (not the teach pendant)
- Log in as an Expert or Administrator. For KR C4: KUKA Menu >Configuration >User Group. Default password is kuka
- Minimize the "HMI". For KR C4:KUKA Menu >Startup >Service >Minimize HMI
- Copy kukavarproxy folder to the Desktop (or anywhere else)
- Start the KUKAVARPROXY.exe
- If you have a problem with the file cswsk32.ocx, Use this command in the Administrator Command Prompt regsvr32.exe c:
  asdf
  cswsk32.ocx while changing the asdf with the true path to the file.
- You can make this program start automatically on reboot by creating a shortcut of KUKAVARPROXY.exe in Windows Start >All Programs>Right click Startup >Open

HMI Network Configuration

- Connect the robot to a network. (private one is recommended) (Port X66 is used)
- Configure the KRC IP. For KR C4: KUKA Menu >Startup >Network Configuration
- Unlock port 7000. For KR C4: KUKA Menu >Startup >Network Configuration >Advanced
- NAT >Add Port >Port number 7000
- Set permitted protocols: tcp/udp

Programming from PC

- Open python (version 2.7 is recommended(
- Import the KRL library in your code
- The default IP for the library is (172.31.1.147), so if you have a different one you should edit this from the library itself.
- The next provided example shows how to use the library functions
- You can edit the library and the KRLDRIVER.SRC to add any additional desired commands.

6.3. Development of Research Applications: Vision System Implementation

6.3.1. Introduction

Robots have been blind for long time resulting to have bad impact on robot's performance and its efficiency. In our modern era of collaborative robotics, however, vision systems are becoming a necessity as we implement robots into more complex jobs and tasks.

Computer vision libraries made it possible to detect human body and hence a safe operation zone can be provided. For more applications, you can have a look at this interesting IEEE article: `http://spectrum.ieee.org/automaton/robotics/diy/top-10-robotic-kinect-hacks`

6.3.2. State of the Art

One of the most important piece of information that a normal camera misses is the depth of the image. The depth is important in recognizing the real world in a proper way. Researchers had found many solutions for this problem like the stereo camera installations, or even including depth sensors with the RGB camera itself, like in the Kinect. Kinect is a depth sensor which is able to return images like an ordinary camera, but instead of color, each pixel value represents the distance to the point. As such, the sensor can be seen as a range- or 3D-camera. For more technical details about Kinect, please refer to this website: `https://ese.wustl.edu/ContentFiles/Research/UndergraduateResearch/CompletedProjects/WebPages/fl12/MattJohnson/kinect1.html` Robotic grasping, object recognition, and human tracking became possible by interfacing Kinect camera to robots, especially robotic manipulators.

6.3.3. Implementations

We've used the Kinect and Robot Operating System (ROS) platform to implement two vision dependent systems on the KUKA robot:

- Visual Servoing System, that makes the robot moves according to the tracked person's hand position
- Safe Operating Zone, where the robot slows down to 10% of its speed when a human is detected in the selected zone.

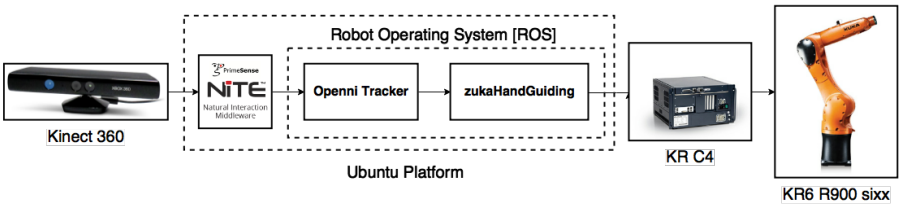
6.3.3.1. Visual Servoing: Hand Guiding

Overview Robotic systems can be controlled using visual data generated by means of computer vision from the workspace to do certain tasks.There are variant approaches to implement a visual servoing, one involves geometric interpretation of information captured by vision sensors, such as estimating the pose of the target and parameters of the data captured by the camera and the motion tracking system.
ZuKa Hand Guiding system allows the human operator to move TCP

(tool center point) along specific path using human skeleton joints. The system can be abstracted to three main layers:

- Top: KUKAVAPROXY, this is a TCP/IP interface to Kuka robots that allows for reading and writing variables and data structures of the controlled manipulators to export target position to the robot in KRL format.
- Middle : Robot Operating System -ROS- which provides communication with all system elements to analyze data captured by kinect.
- Bottom : hardware devices that capture the visual elements of the scene (Kinect XBOX 360)

Figure 6.5.: ZuKa Hand Guide System



Usage

1. KUKA Teach Pendant Setup

    a) While in Expert mode, select **R1/ KRLDRIVER.SRC**

    b) Run the program in the automatic mode.

2. PC Setup

    a) Download **z**ukaHandGuiding

    b) Add the file to Src folder in your package(further details on creating your own package at:
    `http://wiki.ros.org/ROS/Tutorials/CreatingPackage`

    c) Make the node executable

    d) Launch kinect driver “Openni_launch”(refer to Chapter 5 for more details)

    e) Start tracking by running openni_tracker node

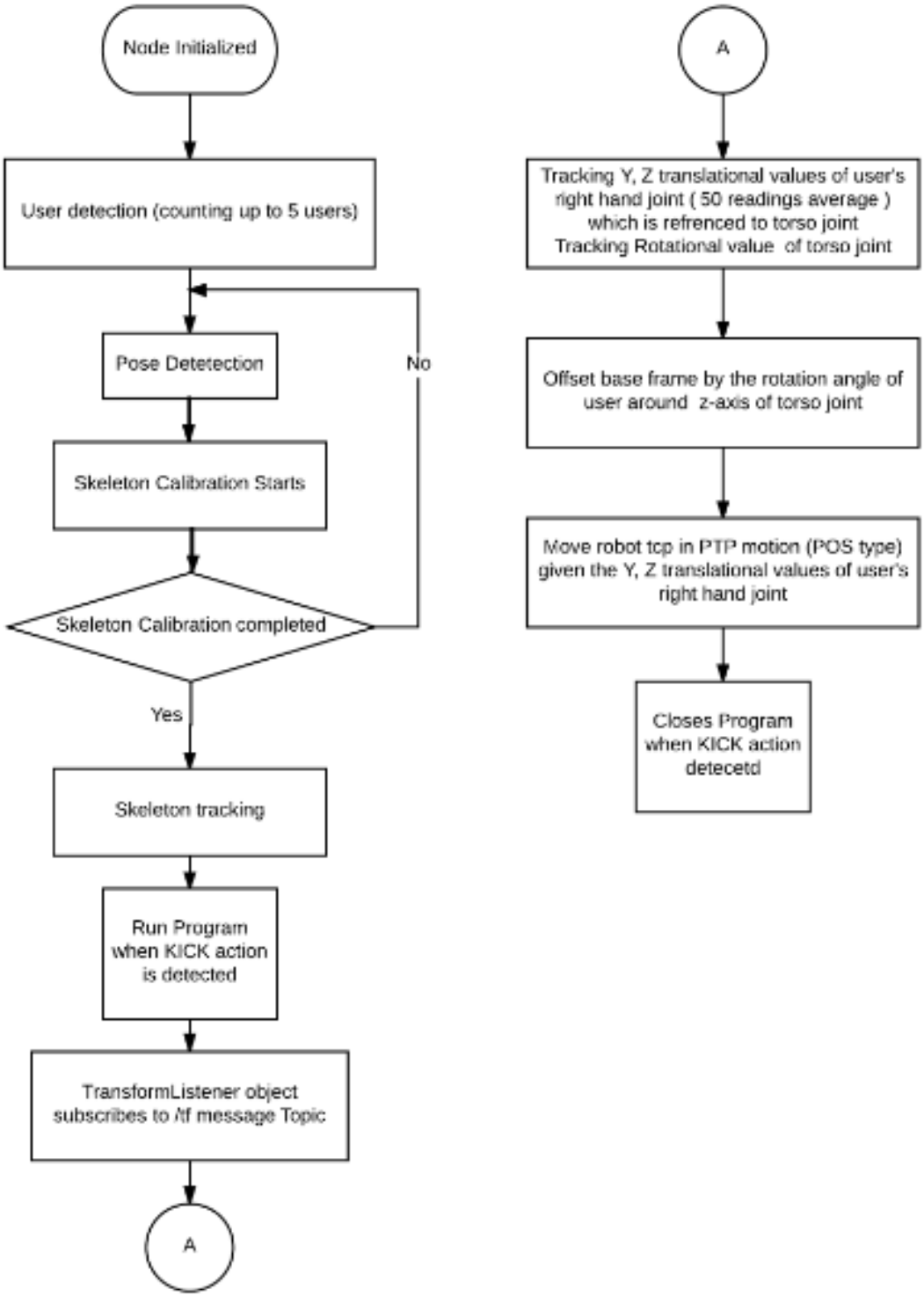
    f) Run **zukaHandGuiding** node

Download Links:

- Openni_tracker
  `https://github.com/mnourgwad/zuka/tree/master/codes/openni_tracker`

- zukaHandGuiding:
  `https://github.com/mnourgwad/zuka/tree/master/codes/zukaHandGuiding`

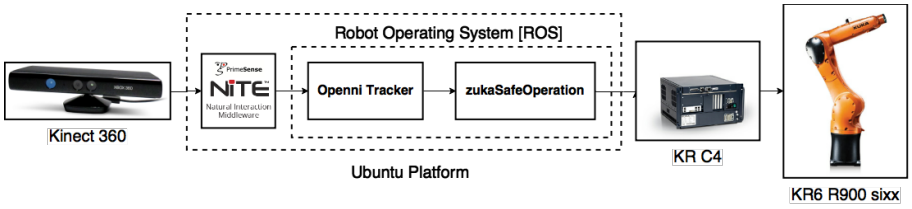
Figure 6.6.: ZuKa Hand Guide System flowchart



6.3.3.2. Safe Operating Zone

Overview Human safety is the main concern which prevents performing some tasks requiring physical interaction between human and robot. Therefore, the safety concept was previously based on eliminating contact between human and robots. Using vision system, we've made it possible for the robot to detect and recognize human body, and its distance to the fixed camera, hence a safe operating zone can be acquired by sending a signal to the robot to change its speed when a human is in the safety zone. This is done by sending the RGB and depth data from the Kinect to the NiTE library, which is a middleware that provides body and skeleton detection, then reading this data by a ROS node to calculate the instantaneous distance of each human's center of mass, and the signal is sent by another ROS node that listens to the stream of the least user distance.

Figure 6.7.: Zuka Safe Operation



Usage Follow these steps for appropriate setup.

- Install kukavarproxy as explained before, and make sure that everything is ok
- Install ROS, NiTE, and openni_tracker as explained before in chapter 5
- Launch the roscore, but don't launch the openni_tracker node
- Copy the our modified openni_tracker.cpp file to your catkin openni_tracker package. Our node publish and extra topic called /closestUser-Distance which gives the least detected distance of any tracked user without the need of standing in the psi calibration position

- Copy the zukaSafeOperation package to your catkin workspace, and run catkin_make to build the node. Don't forget to change node mode to executable if it hasn't.
- Use: rosrun zukaSafeOperation zukaSafeOperation to launch the package
- The package requires a proper internet connection through kukaproxvar, and changes the robot speed to 10% of it's speed when a user is detected within a range of 2 meters

To edit the **detection data and the speed limits**:
In the source code you'll find variables to define the range, and the limited speed. Change these variables to your desired ones

Download Links:

- Openni_tracker
  `https://github.com/mnourgwad/zuka/tree/master/codes/openni_tracker`
- zukaSafeOperation:
  `https://github.com/mnourgwad/zuka/tree/master/codes/zukaSafeOper`

The true function of philosophy is to educate us in the principles of reasoning and not to put an end to further reasoning by the introduction of fixed conclusions.

— George Henry Lewes, (English philosopher and critic of literature, 1817–1878)

Chapter 7
Conclusions and Future Outlook

Robotic milling is a flexible and efficient method aimed at enhancing and developing the industrial sector. Due to the major advancements in all industrial sectors, milling is targeted for further development to achieve higher accuracy, reduce production costs and losses. This project aimed to achieve four goals; accuracy, ease of use, flexibility and safety in the milling process. These goals were achieved over the course of the project with outstanding results, demonstrated in this report. However, the work is still far from over, further developments can be implemented on this project to enhance the obtained results and increase efficiency, level of details obtained, safety and to add further assisting features to the work environment. Possible recommendations include using different G-code generating software other than Mesh CAM, as SprutCAM, power mill and Mastercam, as they offer generation in XYZABC dimensions, enabling G-codes in more than 3-axes, thus more axes for the milling process. Safety in the workspace can be increased by several methods including activating the built-in collision detection in the robot. This feature enabled the robot's links to brake as it detects contact with any external surface. Another feature can be constructing a protective metal cage around the robot to define its workspace and eliminate accidents. In addition to the cage, capacitive sensors can be embedded in the ground around the robot to detect human presence and reduce the speed of the

robot or stop it completely.

Industrial automation is the trend of our age, further research and development work should be targeted at implementing new methods, features and take advantage of the current developments to reach higher targets aimed to increase the prosperity of mankind.

7.1. Project Contributions

The results of the project studies and implementation include, but not limited to;

- The manufacturing of the robot's base, with mathematically calculated data endorsed by CAD studies, contributing in a stable, secure and robust base that can support the weight of the robot and tolerate the forces resulting from the robot's motion without major failure or errors.
- The attachment and operation of a pneumatic gripper, leading to the development and implementation of software tools for drawing and palletizing.
- The development of different software tools to obtain the appropriate KRL codes used in the milling process.
- The development of a safety system in the robot's workspace, similar to KUKA AG's own Collision detection, which stops thee robot from moving when it hits a solid surface. However, being more efficient and safe, in terms that it does not require actual contact or collision but significantly reduced the operation speed of the robot when someone enters a defined perimeter of the robot's workspace. This is achieved using a Microsoft Kinect device for obtaining visual input of the workspace.

The results of the work exceeded both the preset expectations and goals for the project, resulting in a wide variety of applications and an extension in our own knowledge base, which is the most important achievement.

Chapter A
Appendices

A.1. List of Symbols and Abbreviations

- Repeatability - variability in returning to the same previously taught position/configuration
- Accuracy - variability in moving to a target in space that has not been previously taught
- Tool speed - linear speed capability when tool moving along a curvilinear path
- Screw speed - rotational speed when tool is being rotated about an axis in space
- Joint interpolated motion - motion where joint taking longest time to make the joint change governs the motion and the other joints are slowed in proportion so that all joints accomplish their joint changes simultaneously with the slowest joint
- Joint limits - either the software or physical hardware limits which constrain the operating range of a joint on a robot. The software limits have a smaller range than the hardware limits.
- Joint speed limits - speed limit for robot joints, which limit how fast the links of a robot may translate or rotate.

- Point-to-point motion - characterized by starting and stopping between configurations or as the tool is moved between targets.
- Continuous path motion - characterized by blending of motion between configurations or targets, usually with the loss of path accuracy at the target transitions, as the robot moves between configurations/targets.
- Interpolation (kinematic) capabilities - robot usually capable of both forward and inverse kinematics. Both combine to give the robot the capability to move in joint space and in Cartesian space. We typically refer to the moves as joint, linear, or circular interpolation.
- Forward kinematics - specifying the joint values to accomplish a robot move to a new configuration in space. These may not be simple as it seems because secondary joints such as four-bar linkages, ball screws, etc. may be required to accomplish this motion.
- Inverse kinematics - solving a mathematical model of the robot kinematics to determine the necessary joint values to move the tool to a desired target (frame) in space. This is accomplished by frame representation whereby a triad (xyz axes) is attached to the tool on the robot and a target frame is attached to the part or operating point in the workcell. The inverse kinematics determine the joint values that align the tool triad with the target triad.
- I/O - input/output which consist of ON/OFF signal values, threshold values, or analog signal values which allow the control of or response to external devices/sensors as required to sequence workcell operations.
- Programming language - The language and logical constructs used to program the set of operational instructions used to control robot movement and interact with sensors and other cell devices.
- Multi-tasking - ability to process more than one program at a time or process I/O concurrently.
- Load capability - force and torque capability of the robot at its tool interface
- TCF - tool or terminal control frame

- TCP - tool/terminal control point
- Teach Pendant - Operator interface device used to teach/save robot
- configurations and program simple instructions.

A.2. Kinect specifications

Sensor:

- Colour and depth-sensing lenses
- Voice microphone array
- Tilt motor for sensor adjustment

Field of View:

- Horizontal field of view: 57 degrees
- Vertical field of view: 43 degrees
- Physical tilt range: 27 degrees
- Depth sensor range: 1.2m - 3.5m

Data Streams:

- 320x240 16-bit depth @ 30 frames/sec
- 640x480 32-bit colour @ 30 frames/sec
- 16-bit audio @ 16 kHz

Skeletal Tracking System:

- Tracks up to 6 people, including 2 active players
- Tracks 20 joints per active player

The interaction space is the area in front of the Kinect sensor where the infrared and color sensors have an sensor. If the lighting is not too bright and not too dim, and the objects being tracked are not too reflective skeletons. While a sensor is often placed in front of and at the level of a user's head, it can be placed in a w

The interaction space is defined by the field of view of the Kinect cameras, which is listed in Kinect for Wind To increase the possible interaction space, tilt the sensor using the built-in tilt motor. The tilt motor suppo greatly increases the possible interaction space in front of the sensor.

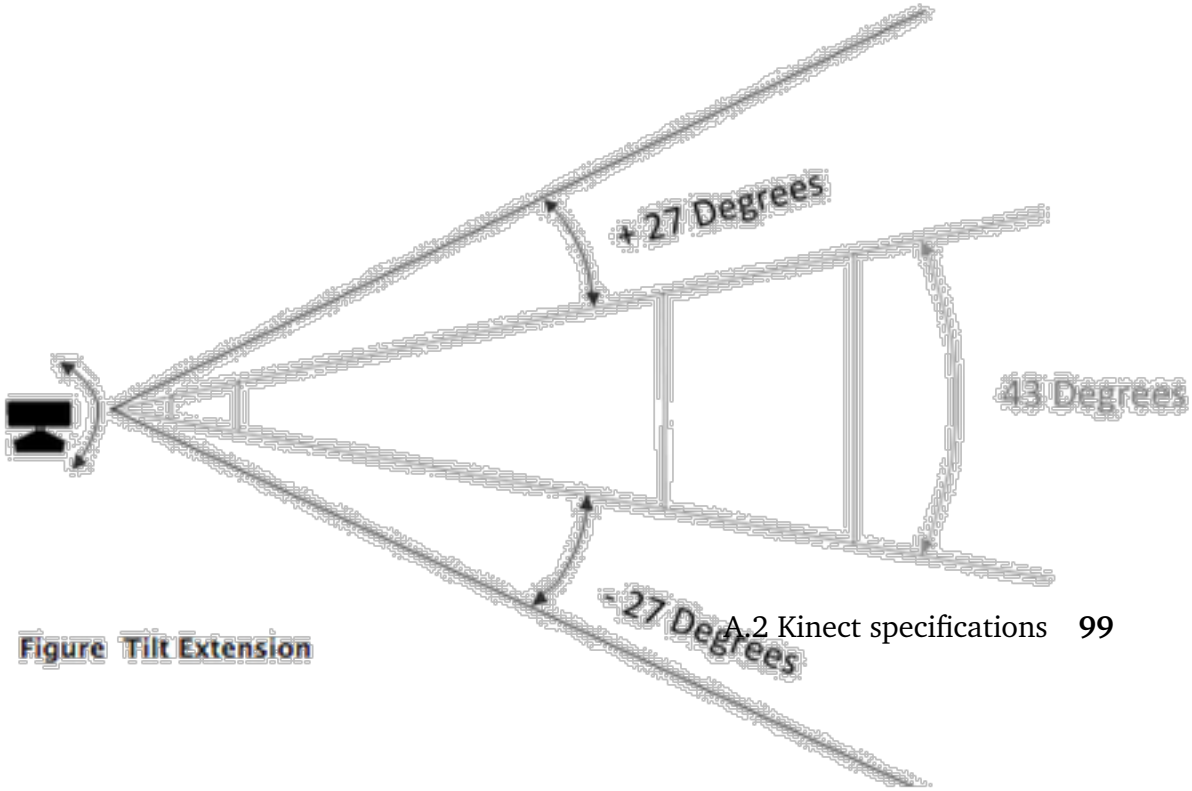


Figure Tilt Extension

Specifications for the Kinect

Kinect	Array Specifications
Viewing angle	43° vertical by 57° horizontal field of view
Vertical tilt range	±27°
Frame rate (depth and color stream)	30 frames per second (FPS)
Audio format	16-kHz, 24-bit mono pulse code modulation (PCM)
Audio input characteristics	A four-microphone array with 24-bit analog-to-digital converter (ADC) and Kinect-resident signal processing including acoustic echo cancellation and noise suppression
Accelerometer characteristics	A 2G/4G/8G accelerometer configured for the 2G range, with a 1° accuracy upper limit.

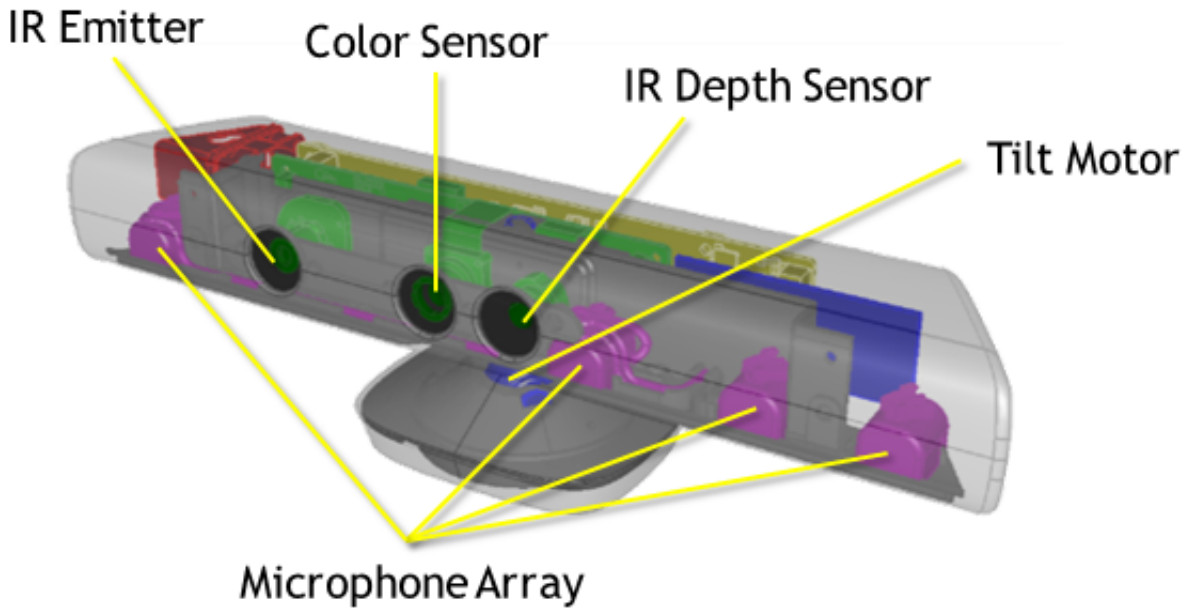


Table 6: Parameters of the robot shown in Fig. 3 [9].

A [mm]	B [mm]	C [mm]	D [mm]	E [mm]	F [mm]	G [mm]	H [mm]	I [mm]	J [mm]
1276	1620	901.5	656	245.5	851.5	420	455	400	855

Table 7: Parameters D-H for the robot KR 6 R900 sixx AGILUS.

i	α_i [rad]	d_i [mm]	a_i [mm]	θ_i [rad]
1	$-\pi/2$	$d_1 = I$	$a_1 = 25$	$q_1 = \pi/2$
2	0	0	$a_2 = H$	$q_2 = -\pi/2$
3	$\pi/2$	0	$a_3 = 35$	$q_3 = 0$
4	$-\pi/2$	$d_4 = -(G-1.4)$	0	$q_4 = 0$
5	$\pi/2$	0	0	$q_5 = 0$
6	π	$d_6 = -80$	0	$q_6 = 0$

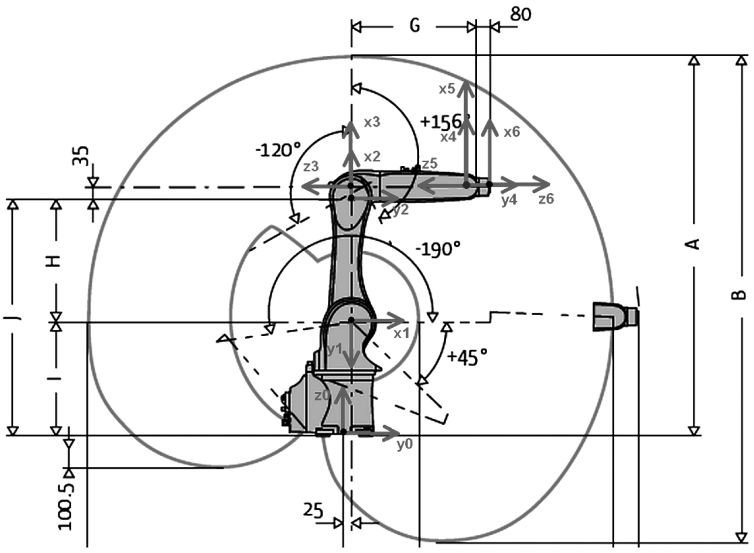
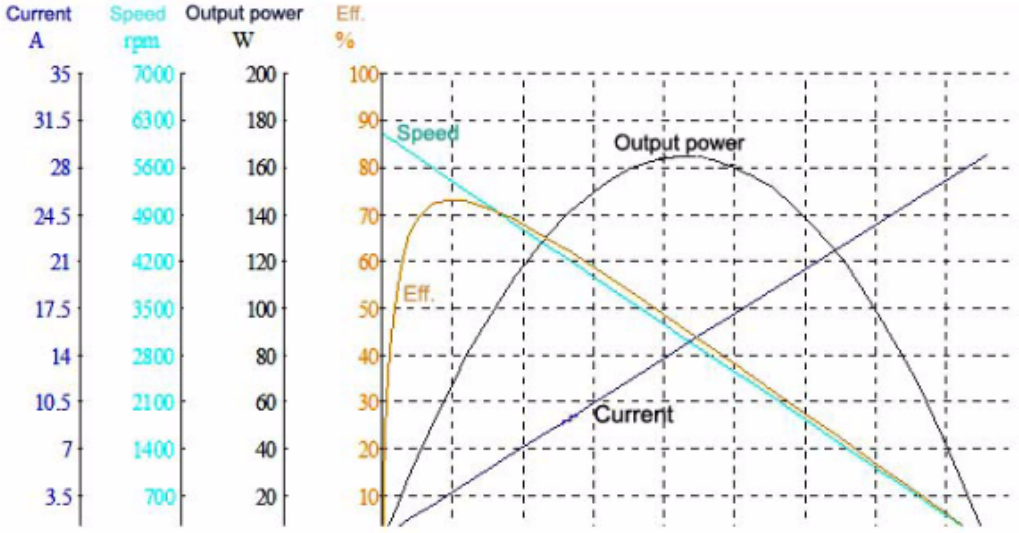


Figure 3: Dimensions of the robot KR 6 R900 sixx AGILUS [9] with marked coordinate systems.

1). 300W Spindle Test Sheet



Characteristic	Voltage	Current	Input power	Torque	Speed	Output power	Efficiency
	V	A	W	mN.m	rpm	W	%
No._Load	24.19	0.504	12.19	0.0	6109	0.000	0.0
Eff_Max	24.28	4.001	97.12	126.7	5359	71.09	73.2
Output power_Max	24.52	14.22	348.6	496.8	3168	164.8	47.3
Torque_Max	24.88	28.99	721.4	1032.2	0	0.000	0.0
End	24.88	28.99	721.4	1032.2	0	0.000	0.0

2. Spindle brush life-span(Max.): 1000 hours @ unload 48V

3. Shaft run-out: 0.02 ~0.05

A.3. Codes

A.3.1. KUKAVARPROXY.py

```
import socket                                                    #
    Used for TCP/IP communication
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
           # Initializing client connection

class KUKA(object):

    def __init__(self, TCP_IP):
        try:
            client.connect((TCP_IP, 7000))                       #
               Open socket. kukavarproxy actively listens on TCP
               port 7000
        except:
            self.error_list(1)


    def send (self, var, val, msgID):
        """
        kukavarproxy message format is
        msg ID in HEX                       2 bytes
        msg length in HEX                   2 bytes
        read (0) or write (1)               1 byte
        variable name length in HEX         2 bytes
        variable name in ASCII              # bytes
        variable value length in HEX        2 bytes
        variable value in ASCII             # bytes
        """
        try:
            msg = bytearray()
            temp = bytearray()
            if val != "":
                val = str(val)
                msg.append((len(val) & 0xff00) >> 8)             #
                   MSB of variable value length
                msg.append((len(val) & 0x00ff))                  #
                   LSB of variable value length
                msg.extend(map(ord, val))                        #
                   Variable value in ASCII
            temp.append(bool(val))                               #
               Read (0) or Write (1)
            temp.append(((len(var)) & 0xff00) >> 8)              #
               MSB of variable name length
            temp.append((len(var)) & 0x00ff)                     #
               LSB of variable name length
            temp.extend(map(ord, var))                           #
               Variable name in ASCII
            msg = temp + msg
```

```
            del temp[:]
            temp.append((msgID & 0xff00) >> 8)                    #
               MSB of message ID
            temp.append(msgID & 0x00ff)                           #
               LSB of message ID
            temp.append((len(msg) & 0xff00) >> 8)                 #
               MSB of message length
            temp.append((len(msg) & 0x00ff))                      #
               LSB of message length
            msg = temp + msg
        except :
            self.error_list(2)
        try:
            client.send(msg)
            return  client.recv(1024)                             #
               Return response with buffer size of 1024 bytes
        except :
            self.error_list(1)


    def __get_var(self, msg):
        """
        kukavarproxy response format is
        msg ID in HEX                       2 bytes
        msg length in HEX                   2 bytes
        read (0) or write (1)               1 byte
        variable value length in HEX        2 bytes
        variable value in ASCII             # bytes
        Not sure if the following bytes contain the client number,
           or they're just check bytes. I'll check later.
        """
        try:
            # Python 2.x
            lsb = (int (str(msg[5]).encode('hex'),16))
            msb = (int (str(msg[6]).encode('hex'),16))
            lenValue = (lsb <<8 | msb)
            return msg [7: 7+lenValue]

            """
            # Python 3.x
            lsb = int( msg[5])
            msb = int( msg[6])
            lenValue = (lsb <<8 | msb)
            return str(msg [7: 7+lenValue],'utf-8')
            """

        except:
            self.error_list(2)

    def read (self, var, msgID=0):
        try:
            return self.__get_var(self.send(var,"",msgID))
```

```
        except :
            self.error_list(2)


    def write (self, var, val, msgID=0):
        try:
            if val != (""): return self.__get_var(self.send(var,val
                ,msgID))
            else: raise self.error_list(3)
        except :
            self.error_list(2)


    def disconnect (self):
            client.close()                                          #
                CLose socket


    def error_list (self, ID):
        if ID == 1:
            print ("Network Error (tcp_error)")
            print ("    Check your KRC's IP address on the network,
                and make sure kukaproxyvar is running.")
            self.disconnect()
            raise SystemExit
        elif ID == 2:
            print ("Python Error.")
            print ("    Check the code and uncomment the lines
               related to your python version.")
            self.disconnect()
            raise SystemExit
        elif ID == 3:
            print ("Error in write() statement.")
            print ("    Variable value is not defined.")
```

A.3.2. KRLDRIVER.src

```
&ACCESS RVP
&REL 8
&PARAM EDITMASK = *
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
DEF KRLDRIVER( )
;FOLD INI;%{PE}
  ;FOLD BASISTECH INI
    GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
    INTERRUPT ON 3
    BAS (#INITMOV,0 )
  ;ENDFOLD (BASISTECH INI)
;FOLD SPOTTECH INI
USERSPOT(#INIT)
;ENDFOLD (SPOTTECH INI)
;FOLD GRIPPERTECH INI
USER_GRP(0,DUMMY,DUMMY,GDEFAULT)
;ENDFOLD (GRIPPERTECH INI)
  ;FOLD USER INI
    ;Make your modifications here

  ;ENDFOLD (USER INI)
;ENDFOLD (INI)
KRLD_COM=0
$ADVANCE=1
$PAL_MODE=FALSE
$OV_PRO=100
PTP {A1 0, A2 -90, A3 90, A4 0, A5 0, A6 0}
WHILE TRUE
SWITCH KRLD_COM
CASE 1
KRLD_COM=0
PTP KRLD_AXIS C_PTP

CASE 2
KRLD_COM=0
PTP KRLD_POS C_PTP

CASE 3
KRLD_COM=0
LIN KRLD_POS C_DIS

CASE 4
KRLD_COM=0
CIRC KRLD_POS_AUX, KRLD_POS

CASE 5
KRLD_COM=0
PTP_REL KRLD_AXIS C_PTP

CASE 6
```

```
KRLD_COM=0
PTP_REL KRLD_POS C_PTP

CASE 7
KRLD_COM=0
LIN_REL KRLD_POS C_DIS

CASE 8
KRLD_COM=0
WAIT SEC KRLD_SLEEP

CASE 9
KRLD_COM=0
$BASE=KRLD_BASE:KRLD_BASE_OFFSET

CASE 10
KRLD_COM=0
$TOOL=KRLD_TOOL

CASE 11
KRLD_COM=0
$APO.CPTP=KRLD_APO
$APO.CDIS=KRLD_APO

CASE 12
KRLD_COM=0
$VEL.CP=KRLD_VEL_CP

CASE 13
KRLD_COM=0
BAS(#VEL_PTP,KRLD_VEL_PTP)

CASE 14
KRLD_COM=0
$OUT[KRLD_IO]=KRLD_SIGNAL

CASE 15
KRLD_COM=0
$PAL_MODE= KRLD_SIGNAL
ENDSWITCH
ENDWHILE
END
```

A.3.3. krldriver.py

```
from kukavarproxy import *

robot = KUKA('172.31.1.147')

axes_limits = [
    [-170,170],[-190,45],[-120,156],[-185,185],[-120,120],[-350,350]]

AXIS = 'a'
POS = 'p'
FALSE = 0
TRUE = 1

def INI(VEL_CP_VAL=3, VEL_AXIS_VAL=100, OV_PRO_VAL=10 ,APO_VAL=0):
        VEL_CP(VEL_CP_VAL)
        VEL_PTP(VEL_AXIS_VAL)
        OV_PRO(OV_PRO_VAL)
        APO(APO_VAL)
        BASE(525,0,890,0,0,0)
        TOOL(0,0,0,0,-90,0)
        PTP(POS,0,0,0,0,0,0)

def PTP_HOME():
        PTP(AXIS,0,-90,90,0,0,0)

def GRIPPER_OPEN():
        OUT(1,TRUE)
        OUT(1,FALSE)
        OUT(4,TRUE)
        OUT(4,FALSE)

def GRIPPER_CLOSE():
        OUT(4,TRUE)
        OUT(4,FALSE)
        OUT(1,TRUE)
        OUT(1,FALSE)

def OUT(ioPort, Signal):
        while (int (robot.read("KRLD_COM"))): continue
        robot.write("KRLD_IO",ioPort)
        if Signal == True: robot.write("KRLD_SIGNAL","TRUE")
        else: robot.write("KRLD_SIGNAL","FALSE")
        robot.write("KRLD_COM",14)

def PAL_MODE(Signal):
        while (int (robot.read("KRLD_COM"))): continue
        if Signal == True: robot.write("KRLD_SIGNAL","TRUE")
        else: robot.write("KRLD_SIGNAL","FALSE")
        robot.write("KRLD_COM",15)

def PTP(interpolation, v1="",v2="",v3="",v4="",v5="",v6=""):
```

```
        if v1 == v2 == v3 == v4 == v5 == v6 == "":
                raise error(2)
                return 0
        if interpolation == AXIS:
                axes = [v1,v2,v3,v4,v5,v5]
                for index,axis in enumerate(axes):
                        pass
                        #if float(axis) <= axes_limits[index][0] or
                            float(axis) >= axes_limits[index][1]:
                           raise error(1,(index+1))
                while (int (robot.read("KRLD_COM"))): continue
                destination = format_axis(v1,v2,v3,v4,v5,v6)
                robot.write("KRLD_AXIS", destination)
                robot.write("KRLD_COM",1)
        elif interpolation == POS:
                while (int (robot.read("KRLD_COM"))): continue
                destination = format_pos(v1,v2,v3,v4,v5,v6)
                robot.write("KRLD_POS",destination)
                robot.write("KRLD_COM",2)
        else:
                raise error(3)

def PTP_REL(interpolation, v1="",v2="",v3="",v4="",v5="",v6=""):
        if v1 == v2 == v3 == v4 == v5 == v6 == "":
                raise error(2)
                return 0
        if interpolation == AXIS:
                while (int (robot.read("KRLD_COM"))): continue
                destination = format_axis(v1,v2,v3,v4,v5,v6)
                robot.write("KRLD_AXIS", destination)
                robot.write("KRLD_COM",5)
        elif interpolation == POS:
                while (int (robot.read("KRLD_COM"))): continue
                destination = format_pos(v1,v2,v3,v4,v5,v6)
                robot.write("KRLD_POS",destination)
                robot.write("KRLD_COM",6)
        else:
                raise error(3)

def LIN (x="",y="",z="",a="",b="",c=""):
        if x == y == z == a == b == c == "": raise error(3)
        while (int (robot.read("KRLD_COM"))): continue
        destination = format_pos(x,y,z,a,b,c)
        robot.write("KRLD_POS",destination)
        robot.write("KRLD_COM",3)

def LIN_REL (x="",y="",z="",a="",b="",c=""):
        if x == y == z == a == b == c == "": raise error(3)
        while (int (robot.read("KRLD_COM"))): continue
        destination = format_pos(x,y,z,a,b,c)
        robot.write("KRLD_POS",destination)
        robot.write("KRLD_COM",7)
```

```
def CIRC (x="",y="",z="",a="",b="",c="",xAux="",yAux="",zAux="",
    aAux="",bAux="",cAux=""):
        if x == y == z == a == b == c == "": raise error(3)
        if xAux == yAux == zAux == aAux == bAux == cAux == "":
            raise error(5)
        while (int (robot.read("KRLD_COM"))): continue
        destination = format_pos(x,y,z,a,b,c)
        robot.write("KRLD_POS",destination)
        auxilliary = format_pos(xAux,yAux,zAux,aAux,bAux,cAux)
        robot.write("KRLD_POS",auxilliary)
        robot.write("KRLD_COM",4)

def WAIT(time_sec):
        while (int (robot.read("KRLD_COM"))): continue
        robot.write("KRLD_SLEEP", abs(time_sec))
        robot.write("KRLD_COM",8)

def BASE(x=0,y=0,z=0,a=0,b=0,c=0,offX=0,offY=0,offZ=0,offA=0,offB
    =0,offC=0,BASE_DATA=-1):
        while (int (robot.read("KRLD_COM"))): continue
        if BASE_DATA > 0:
                base_frame = "BASE_DATA["  + str(BASE_DATA) + "]"
                robot.write("KRLD_BASE", robot.read(base_frame))
        else:
                if x == y == z == a == b == c == "": raise error(3)
                base_frame = format_pos(x,y,z,a,b,c)
                robot.write("KRLD_BASE", base_frame)
        offset_data = format_pos(offX,offY,offZ,offA,offB,offC)
        robot.write("KRLD_BASE_OFFSET", offset_data)
        robot.write("KRLD_COM",9)

def TOOL(x=0,y=0,z=0,a=0,b=0,c=0,TOOL_DATA=-1):
        while (int (robot.read("KRLD_COM"))): continue
        if TOOL_DATA > 0:
                tool_frame = "TOOL_DATA["  + str(TOOL_DATA) + "]"
                robot.write("KRLD_TOOL", robot.read(tool_frame))
        else:
                if x == y == z == a == b == c == "": raise error(3)
                tool_frame = format_pos(x,y,z,a,b,c)
                robot.write("KRLD_TOOL", tool_frame)
        robot.write("KRLD_COM",10)

def APO(approximation_value):
        while (int (robot.read("KRLD_COM"))): continue
        robot.write("KRLD_APO", approximation_value )
        robot.write("KRLD_COM",11)

def VEL_CP(velocity_value):
        while (int (robot.read("KRLD_COM"))): continue
        robot.write("KRLD_VEL_CP", velocity_value)
        robot.write("KRLD_COM",12)
```

```
def VEL_PTP(velocity_value):
        while (int (robot.read("KRLD_COM"))): continue
        robot.write("KRLD_VEL_PTP", velocity_value)
        robot.write("KRLD_COM",13)

def OV_PRO(percentage):
        while (int (robot.read("KRLD_COM"))): continue
        if percentage < 0 or percentage > 100: raise error(4)
        robot.write("$OV_PRO", percentage)


def format_axis(A1, A2, A3, A4, A5, A6):
        axes = [A1,A2,A3,A4,A5,A6]
        axes_names = ["A1","A2","A3","A4","A5","A6"]
        instruction = " "
        for index,axis in enumerate(axes):
                if axis != "":
                        if instruction[len(instruction)-1] != " ":
                                instruction += ", "
                        instruction += axes_names[index] + " " +
                           str(axis)
        return "{" + instruction + "}"

def format_pos(x, y, z, a, b, c):
        coordinates = [x,y,z,a,b,c]
        coordinates_names = ["X","Y","Z","A","B","C"]
        instruction = " "
        for index,coordinate in enumerate(coordinates):
                if coordinate != "":
                        if instruction[len(instruction)-1] != " ":
                                instruction += ", "
                        instruction += coordinates_names[index] + "
                            " + str(coordinate)
        return "{" + instruction + "}"

def error(index, axis=9):
        if index == 1: print("A" + str(axis)+ " limits violated.")
        if index == 2: print("One parameter- at least- should be
           defined for motion.")
        if index == 3: print("Interpolation type is wrongly defined
           .")
        if index == 4: print("$OV_PRO values should be percentage."
           )
        if index == 5: print("Incorrect base number.")
        if index == 6: print("One auxilliary coordinate- at least-
           should be defined for CIRC motion.")
        raise SystemExit
```

A.3.4. Palletizing Example on KRL Driver

```
from krldriver import *

def take():
        PTP(POS,"","",-857)
        GRIPPER_CLOSE()
        PTP(POS,"","",-757)

def drop():
        PTP(POS,"","",-857)
        GRIPPER_OPEN()
        PTP(POS,"","",-757)

def p(n):
        if n == 1: PTP(POS,-80,-125,-757)
        if n == 2: PTP(POS,-80,50,-757)
        if n == 3: PTP(POS,90,50,-757)
        if n == 4: PTP(POS,90,-125,-757)

distenation = 1
source = 4

VEL_CP(3)
VEL_PTP(100)
OV_PRO(10)
APO(0)
BASE(525,0,890,0,0,0)
TOOL(0,0,0,0,-90,0)
PAL_MODE(TRUE)

while 1:
        p(source)
        take()
        p(distenation)
        drop()
        distenation -=1
        source -=1
        if distenation == 0: distenation = 4
        if source == 0: source = 4

"""
EXAMPLES

Setup Functions
        TOOL(0,0,0,0,0,0)                                        #
           or    TOOL(TOOL_DATA=3)
        BASE(525,0,890,0,0,0)                              # or
           BASE(BASE_DATA=30)
        APO(1.5)
                  # Approximation value of 1.5mm for C_PTP and
           C_DIS values
```

```
        VEL_CP(.15)
                    # Sets CP motions' velocity (LIN and CIRC) to
           .15m/s
        VEL_PTP(30)
                    # Sets axis velocities at PTP motion to 30% of
           the maximum allowable velocity
        OV_PRO(50)
                    # Sets the overall program velocities (VEL_CP
           and VEL_PTP) to 50% of their values

Motion Functions
        PTP(AXIS,0,-90,90,80,70,60)                      # or    PTP
           (POS,0,0,60,0,0,0)
        PTP_REL(POS,"","",20)                            # Unchanged
            values of X, Y, A, B, and C
        LIN(0,10)
        LIN_REL("",10)
        CIRC("",10,"","","","",-5,5)            # If you're at the
           (0,0,0) point, this line will draw half a circle [or
           letter C]
Other
        OUT(1,TRUE)                                                #
           Sets output port 1 to TRUE signal
        WAIT(.2)
                    # Wait for 200ms

Auxilliary Functions [Not an original part of KRL]
        PTP_HOME()
                    # Robot's home position
        GRIPPER_OPEN()                                             #
           Does the required instruction to operate a gripper
           actuated by port 1 and 4
        GRIPPER_CLOSE()
        INI()
                    # Sets  VEL_CP = 3m/s
   VEL_PTP = 100%
        OV_PRO = 10%
        APO o= 0
        TOOL = (0,0,0,0,0,0)
        BASE = (525,0,890,0,0,0)  [The same position for home of
           KR6 r900 sixx]
  and goes to the PTP(POS,0,0,0,0,0,0) of the Base to initialize
     the $POS_ACT variable to be
  able to use PTP_REL and LIN_REL motions.

  It's recommended to use this function at the beginning.
  You can change the values of the speeds and approximation by
  passing parameters to this function
  INI(VEL_CP, VEL_PTP, OV_PRO, APO)
    """
```

References

1. http://www.mmsonline.com/articles/a-new-milling-101-what-milling-is-then-and-now-plus-a-glossary-of-milling-terms
2. http://www.mmsonline.com/articles/a-new-milling-101-what-milling-is-then-and-now-plus-a-glossary-of-milling-terms
3. http://www.sickinsight-online.com/safety-and-more-sick-provides-protection-and-navigation-data-for-kukas-kmr-iiwa/
4. http://medicaldesign.com/contract-manufacturing/modern-cnc-machining-prescription-product-development
5. http://articles.sae.org/11272/
6. https://en.wikipedia.org/wiki/Multiaxis_machining
7. https://en.wikipedia.org/wiki/Milling_(machining)
8. http://www.engineersrule.com/motion-studies-and-how-to-do-them/
9. http://help.solidworks.com/2017/English/SolidWorks/cworks/c_Study_Types.htm
10. SolidWorks forums
11. Dimensions manual
12. Specification manual
13. https://robotics.stackexchange.com/questions/10256/dynamic-torque-simulation-for-a-6-dof-robotic-arm

14. Handbook of Industrial Robotics, Volume 1, edited by Shimon Y. Nof.
15. KUKA manual “01-Mastering and unmastering”
16. KUKA manual “07-KSS_82_Software programming_en”
17. KUKA manual “KST_WorkVisual_en”
18. KUKA manual “KUKA.Sim_2.2_Installation_en”
19. KUKA manuals “KST_GripperSpotTech_32_en”
20. http://blog.robotiq.com/bid/72815/Top-5-Applications-for-Robotic-Electric-Grippers
21. https://www.kuka.com/en-de/products/robot-systems/software/application-software/kuka_gripper_spottech
22. https://www.robot-forum.com/robotforum/kuka-robot-forum/valves-and-inputs-on-the-wrist-krc4-agilus-kr6-r900-sixx-17168/
23. https://www.aliexpress.com/item/0-3kw-Air-cooled-spindle-ER11-chuck-CNC-300W-Spindle-Motor-Power-Supply-speed-governor-Clamp/326989
24. https://www.repetier.com/repetier-g-code-plugin-for-inskscape/
25. http://javakuka.com/xyzabc/
26. KST Expert Programming Manual KSS 5.2
27. INTRODUCING ROBOTICS: MAKING ROBOTS MOVE QUEENSLAND UNIVERSITY OF TECHNOLOGY
28. Robotics, Vision and Control: Fundamental Algorithms in MATLAB. Book by Peter Corke
29. Shigley’s Mechanical Engineering Design, Richard G. Budynas, J. Keith Nisbett, 10th edition, 2014.
30. KR AGILUS six with W & C variants Specifications.
31. http://wiki.ros.org/ROS/Introduction
32. A Gentle Introduction to ROS, Jason M. O’Kane
33. Learning ROS for Robotics Programming, Aaron Martinez ,Enrique Fernández.

34. OpenNI user Guide.
35. https://rosresearch.wordpress.com/2013/12/26/openni-nite-skeleton-tracking-algorithm-related-notes/
36. http://pr.cs.cornell.edu/humanactivities/data/NITE.pdf
37. http://wordpress.mrreid.org/2011/08/20/kinect-physics/
38. http://sourceforge.net/projects/openshowvar/
39. https://github.com/aauc-mechlab/jopenshowvar/
40. http://filipposanfilippo.inspitivity.com/publications/jopenshowvar-an-open-source-cross-platform-communication-interface-to-kuka-robots.pdf
41. https://www.robodk.com/forum/attachment.php?aid=4
42. http://answers.ros.org/question/30575/detecting-user-position-with-kinect-without-psi-pose/
43. KSS KUKA Robot Programming 1 Training -KSS8_v4.
44. KST-Expert_Programming_manual KSS 5.2.
45. Notes_ Bochum KUKA Programming.
46. KRL Quickguide Syntax 8x.
47. 07-KSS_82_Software programming.
48. KST_GripperSpotTech_32.
49. Spez_KR_AGILUS_sixx_CR.
50. http://javakuka.com/xyzabc
51. www.globalrobots.com

Bibliography

[Byrd and de Vries, 1990] Byrd, J. and de Vries, K. A. (1990). six-legged telerobot for nuclear applications development. *Int. J. Robot*, 9:43–52. []

[Cousins, 2011] Cousins, S. (2011). Exponential growth of ros [ros topics]. *IEEE Robotics Automation Magazine*, 18(1):19–20. []

[Digia, 2017] Digia (2017). Qt project. `http://qt-project.org`. []

[Ding et al., 2010] Ding, X., Rovetta, A., Zhu, J. M., and Wang, Z. (2010). Locomotion analysis of hexapod robot. *INTECH Open Access Publishe*. []

[Dynamics, 2015a] Dynamics, B. (2015a). Dedicated to the science and art of how things move, Boston dynamics. []

[Dynamics, 2015b] Dynamics, B. (2015b). *Boston Dynamics*. Boston dynamics. []

[Dürr et al., 2004] Dürr, V., Schmitz, J., and Cruse, H. (2004). "behaviour-based modeling of hexapod locomotion: linking biology and technical application". *Arthropod Structure & Development*, 33:237–250. []

[G.M.Nelson, 1997] G.M.Nelson, R. (1997). Design & simulation of a crockroach like-hexapod robot. In *the IEEE international conference on Robotics & automation*, New Mexico. IEEE. []

[Gurfinkel et al.,] Gurfinkel, V., Gurfinkel, E., Devjanin, E., Efremov, E., Zhicharev, D., Lensky, A., Schneider, A., and Shtilman, L. I. o. r. In six-legged walking model of vehicle with supervisory control; nauka press: Moscow, russia, 1982;. *p*, pages 98–147. []

[KanYoneda, 2007] KanYoneda (2007). Light weight quadruped with nine actuators. *journal of robotics & mechatronics*, 19(2). []

[Lewinger and MartinReekie, 2011] Lewinger, W. A. and MartinReekie, H. (2011). A hexapod robot modeled on the stick insect carausius-morosus. In *the 15th international conference on advanced robotics*, Tallinn. []

[Lewinger and Quinn, 2010] Lewinger, W. A. and Quinn, R. D. (2010). A hexapod walks over irregular terrain using a controller adapted from an insects nervous system. In *the IEEE/RSJ international conference on intelligent robots & systems (IROS)*, pages 18–22, Taiwan. IEEE/RSJ. []

[Manoiu-Olaru et al., 2011] Manoiu-Olaru, S., Nitulescu, M., and Stoian, V. (2011). Hexapod robot. mathematical support for modeling and control. In *System Theory, Control, and Computing (ICSTCC), 15th International Conference on*, pages 1–6. []

[McGhee, 1977] McGhee, R. (1977). Control of legged locomotion systems. *In Proceedings of the*, 18:205–215. []

[Mehdigholi&SaeedAkbarnejad, 2012] Mehdigholi&SaeedAkbarnejad, H. (2012). " optimization of watt's six-bar linkage to generate straight& parallel leg motion" in the journal of humanoids. *ISSN*, pages 1996–7209. []

[MohdDaud and KenzoNonami, 2012] MohdDaud and KenzoNonami (2012). Autonomous walking over obstacles by means of lrf for hexapod robot comet-iv. *robotics & Mechatronics*, 24(1). []

[Moore and Buehler, 2001] Moore, E. Z. and Buehler, M. (2001). Stable stair climbing in a simple hexapod robot. Technical report, DTIC Document. []

[Okhotsimski and Platonov, 1973] Okhotsimski, D. and Platonov, A. (1973). Control algorithm of the walking climbing over obstacles.

In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, CA, USA, 20. Stanford. []

[Peternella and Salinari, 1973] Peternella, M. and Salinari, S. (1973). Simulation by digital computer of walking machine control system. In Genova, I., editor, *Proceedings of the 5th IFAC Symposium on Automatic Control in Space of the Conference*. []

[Saranlı, 2002] Saranlı, U. (2002). *Dynamic locomotion with a hexapod robot*. PhD thesis, The University of Michigan. []

[Schneider and Schmucker, 2006] Schneider, A. and Schmucker, U. (2006). Force sensing for multi-legged walking robots: Theory and experiments part 1: Overview and force sensing. In Intelligence, M. and Buchli, J., editors, *Mobile Robotics*, pages 125–174. Germany; Austria, ; Pro Literatur Verlag ARS. []

[Seljanko, 2011] Seljanko, F. (2011). "hexapod walking robot gait generation using genetic gravitational hybrid algorithm" in the 15th international conference on advanced robotics, estonia, june 20-23. 2011. []

[Tedeschi and Carbone, 2014] Tedeschi, F. and Carbone, G. (2014). Designissues for hexapod walking robots. *Robotics*, 3(2):181–206. []

[terrain hex-limbed extra-terrestrial explorer,] terrain hex-limbed extra-terrestrial explorer, N. A. 2009. []