VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Industrial Robot Navigation System Using a Torque Sensor

Systém navigace průmyslového robotu s využitím torque senzoru

Aswin Kumar Sekar

Diploma Thesis

Supervisor: Ing. Radim Hercík, Ph.D

Ostrava, 2024

VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Ref. no.VSB/23/126177
vsboee2303DDA5

Diploma Thesis Assignment

Student: **Aswin Kumar Sekar**

Study Programme: N0714A150002 Control and Information Systems

Title: Industrial Robot Navigation System Using a Torque Sensor
Systém navigace průmyslového robotu s využitím torque senzoru

The thesis language: English

Description:

The thesis deals with the possibilities of navigating the arm of an industrial robot using a torque sensor so that the arm can be navigated by simply guiding the hand. The aim of the thesis is to design and implement software in PCL and an industrial robot to enable this.

1. Introduction to the industrial robot and torque sensor.
2. Analysis of the possibilities of navigating the arm of an industrial robot using a torque sensor.
3. Design of an industrial robot navigation system using a torque sensor.
4. Implementation of an industrial robot navigation system using a torque sensor.
5. Functional verification and testing.
6. Summary and conclusion.

References:

[1] GILCHRIST, Alasdair. Industry 4.0: the industrial internet of things. New York, NY: Springer Science Business Media, [2016]. ISBN 978-1-4842-2046-7.
[2] COLESTOCK, Harry. Industrial Robotics: Selection, Design, and Maintenance. University of Michogen: McGraw-Hill, 2005. ISBN 97-8007-144-052-3.
[3] Kuka: Operational documentation

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **Ing. Radim Hercík, Ph.D.**

Date of issue: 01.09.2023

Date of submission: 30.04.2024

Study programme guarantor: prof. Ing. Jiří Koziorek, Ph.D.

In IS EDISON assigned: 15.12.2023 13:43:44

Abstrakt

Tato práce podrobně popisuje návrh, implementaci a testování nového navigačního systému pro průmyslový robot KUKA KR4, rozšířeného o integraci snímače síly/kroutícího momentu (F/T). Systém umožňuje intuitivní ručně řízenou navigaci ramene robota, což představuje významný pokrok v technologii robotického řízení. Navigační systém zachycuje data o síle v reálném čase ze senzoru F/T připojeného k PC přes Ethernet, což usnadňuje bezproblémový sběr a zpracování dat. Zpracovaná data slouží k aktualizaci koncových souřadnic v proměnné v rámci pracovního prostoru robota, která je řízena pomocí programu napsaného v Kuka Robot Language (KRL). To umožňuje přesné, ruční ovládání pohybů robota, simuluje přirozenější interakci mezi lidmi a roboty a zvyšuje použitelnost v různých průmyslových prostředích. Rozsáhlé testování ověřilo efektivitu a spolehlivost systému a prokázalo schopnost robota provádět složité navigační úkoly se zvýšenou přesností a zkrácenou dobou nastavení. Tato úspěšná implementace nejen dokazuje proveditelnost použití snímačů točivého momentu pro robotickou navigaci v reálném čase, ale také pokládá základy pro budoucí inovace v technologiích interakce mezi robotem a člověkem a nabízí pohled na transformační potenciál F/T snímačů v průmyslové robotice.

Klíčová slova

Navigace průmyslového robota; snímač síly točivého momentu; ruční vedení; KUKA robot

Abstract

This thesis details the design, implementation, and testing of a novel navigation system for the KUKA KR4 industrial robot, enhanced by integrating a force/torque (F/T) sensor. The system enables intuitive hand-guided navigation of the robot arm, representing a significant advancement in robotic control technology. The navigation system captures real-time force data from the F/T sensor connected to a PC via Ethernet, facilitating seamless data acquisition and processing. The processed data is used to update end coordinates in a variable within the robot's workspace, which is controlled using a program written in Kuka Robot Language (KRL). This allows for precise, manual control of the robot's movements, simulating a more natural interaction between humans and robots and increasing usability in various industrial settings. Extensive testing verified the system's effectiveness and reliability, demonstrating the robot's ability to perform complex navigation tasks with enhanced accuracy and reduced setup time. This successful implementation not only proves the feasibility of using torque sensors for real-time robotic navigation but also lays the foundation for future innovations in robot-human interaction technologies, offering insights into the transformative potential of F/T sensors in industrial robotics.

Keywords

Industrial Robot Navigation; Force Torque Sensor; Hand guidance; KUKA robot

Acknowledgement

I would like to thank all those who helped me during the completion of this work. A special thank you to Ing. Radim Hercík, Ph.D. and Bc. Adam Bátrla, whose assistance were particularly crucial. I would also like to acknowledge the assistance provided by artificial intelligence tools, which were useful in improving the clarity of writing.

Contents

List of symbols and abbreviations

KUKA	–	Keller und Knappich Augsburg
FT	–	Force-Torque
ROS	–	Robot Operating System
KSS	–	KUKA System Software
KRL	–	Kuka Robot Language
OEM	–	Original Equiment Manufacturer
UDP	–	User Datagram Protocol
UI	–	User Interface
L/A	–	Link/Activity
LED	–	Light-emitting diode
UART	–	(Universal Asynchronous Receiver/Transmitter
ATI	–	Array Technologies Incorporated.
RDT	–	Raw Data Transfer
OS	–	Operating System
IPv4	–	Internet Protocol Version 4
HTTP	–	Hypertext Transfer Protocol

List of Figures

List of Tables

Chapter 1

Introduction

The utilization of industrial robots across the globe has been consistently escalating each year[1]. Industrial robots have significantly evolved over the past decades, becoming central to modern manufacturing processes due to their efficiency and precision. And here are various navigation methods in robot, and hand-guidance in robotics is one such method, where the operators directly interact with a robot to teach it tasks or guide its movements. This thesis explores the development of an innovative hand-guidance system using a torque sensor to provide a cost-effective and accessible solution for industrial applications.

There are various algorithms for hand-guidance systems that exists, but they need significant investment or cost-effective method needs complex integration therefore limiting their accessibility particularly for small enterprises or academic purposes. There is a need for an alternative that leverages existing hardware and open-source software to provide a more scalable and economically viable solution

The aim of this thesis is to design and implement a hand-guidance system for the KUKA Industrial robot that utilizes a Force Torque sensor and innovative control algorithm that can run in Laptop/PC. The specific goals include:

- Developing a software interface that connects a Laptop to the robot's C3 bridge, enabling real-time robot manipulation and control.
- Implementing control algorithms to facilitate intuitive robot teaching and operation through hand guidance.
- Evaluating the system against key performance metrics including cost-efficiency, integration ease, and user friendliness.

This thesis is intended to provide a viable alternative to high-cost robotic solutions, potentially lowering automation barriers. The use of open-source software and standard hardware aims to

democratize access to sophisticated robotic control technologies. This thesis lays the groundwork for a detailed exploration into developing a new hand-guidance system for industrial robots.

Chapter 2

Analysis of the Navigation System for Industrial Robot

In this chapter we explore the current state of the art in hand-guidance systems for industrial robots, highlighting technological advances, existing challenges, and potential areas for improvement. The analysis is focused on understanding the technologies and methods which are traditionally used and to identify gaps that the current research aims to fill.

Hand-guidance technology has evolved significantly from the time of its creation. It was developed for enhancing the manual control and programming ease, as guiding the robot to a position with hand is simple and fast compared to moving using other control methods. And the Hand-guidance method depends on the force applied and moves to that direction. For this purpose of developing efficient force guidance systems, numerous studies investigate various control methods and sensors. Many modern collaborative systems employ force/torque (F/T) sensors, positioned either in the robot's joints or as external feedback sensors on the end-effector. Alternatively, some systems measure the electrical currents in the motors of the robot to derive force feedback[2].

Sensors integrated into the joints are commonly used in commercially available lightweight collaborative robots. This configuration, as opposed to sensor placement at the robot's end-effector, enhances sensitivity and expands the area within which collisions can be detected. A notable example of research utilizing this approach is found in the study by Abu-Dakka et al. (2015)[3]. In their work, the researchers equipped the KUKA LWR IV collaborative robot with a 3D vision system and implemented a learning by demonstration algorithm to address the challenges of online programming for peg-in-hole tasks. They also assessed the system's performance with and without the learning by demonstration algorithm[3]. It's important to note that they did not include traditional teach pendant methods in their comparative analysis.

Some manufacturers charge extra for the ability to integrate torque sensors into robot joints, and the payload capacity of robots on the market now isn't that large. On the other hand, adding a force/torque (F/T) sensor to the end-effector allows for the implementation of force guidance and

offers a flexible approach. Any industrial robot having a real-time communication connection can use this technique to update its position in response to outside signals[4].

Numerous recent studies have examined force guidance systems utilizing external sensors, which are not initially supplied by the robot manufacturers. The studies from some papers were primarily focus on creating specialized hardware and suggesting new strategies for control and programming. These studies The majority of the control algorithms discussed in these publications are adaptations of established methods such as gravity compensation, virtual tool control, or admittance control. [5][6][7][8].

The solution provided by the robot manufacturer KUKA is the KUKA.ForceTorqueControl system package. With its precise control over the forces and torques delivered throughout various activities, the KUKA.ForceTorqueControl 3.0 improves robotic capabilities. This system's capacity to modify robot movements in response to real-time feedback from process torques and forces makes it suitable for handling complicated production processes. With features like distortion-free positioning and dynamic velocity adjustments along the programmed path, it guarantees compliance with force requirements independent of variations in workpiece size and location. Its capacity to apply force consistently throughout bonding, assembling, roll hemming, and surface finishing procedures like grinding and polishing is critical to its operation[9].

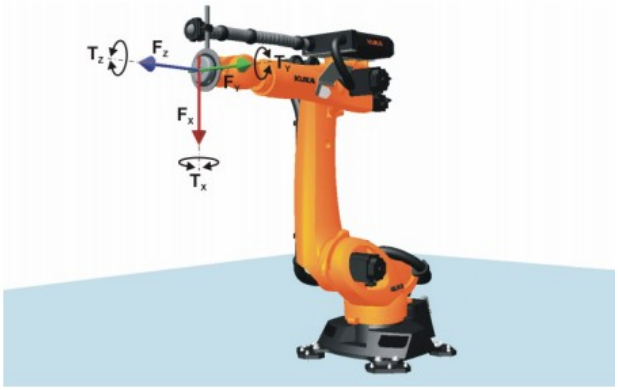

Figure 2.1: Degrees of freedom of force/torque control[9]

In the research by Jonas Loske and Rolf Biesenbach, discussion of enhancement of industrial robots with force-torque sensor. It provides a guide for integrating the force torque sensor into the control system of KUKA robot. This paper outlines the necessary software and hardware modifications required for integrating the KUKA.ForceTorqueControl package with the robot[10].

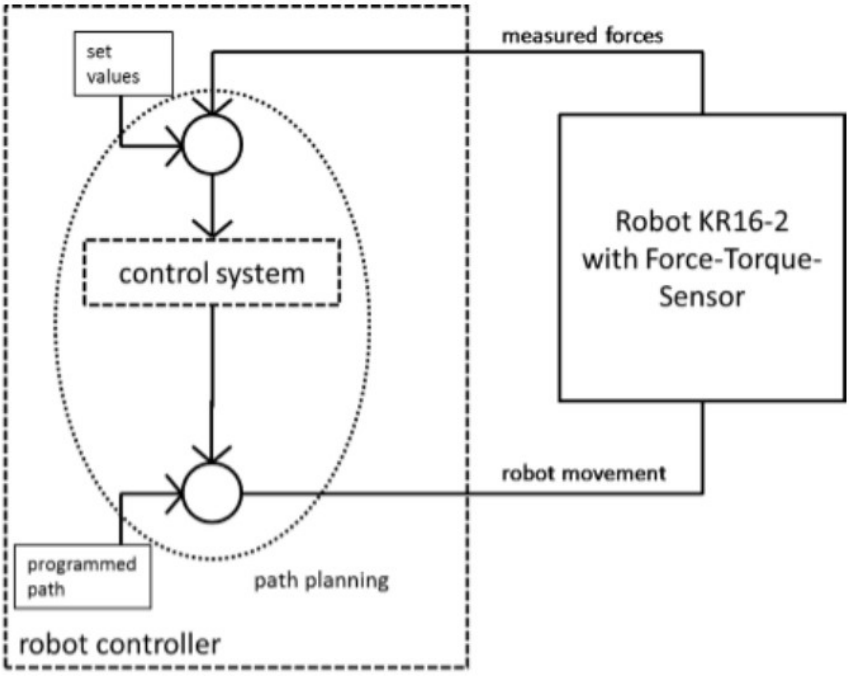

Figure 2.2: Technical Concept of FTCtrl[10]

In the paper titled "Hand Guiding a Virtual Robot Using a Force Sensor," Radovan Gregor, Andrej Babinec, František Duchoň, and Michal Dobiš explore an innovative approach to robot control. Their study introduces a system that can achieve hand guidance movement in robot with the use of the open-source Robot Operating Sysem(ROS). This method uses real-time processing of sensor data to dynamically adjust the robot's trajectory. This study increase the accessibility and flexibility of robotic programming. Although it was not completely successful, this highlights the cost-effective method and potential for more intuitive human-robot collaboration[11].

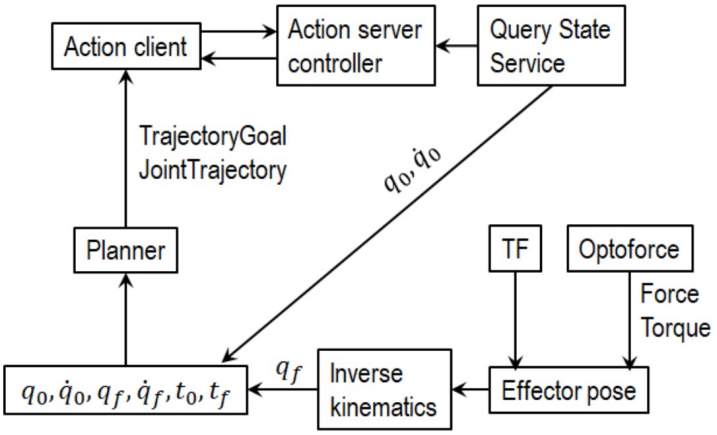

Figure 2.3: Block diagram of system proposed in the paper[11]

After conducting a comprehensive analysis of various navigation methods in industrial robots, especially different algorithms for hand-guidance method,an innovative approach has been developed. In this approach, I have incorporated the use of a external control system which can send information to the robot and the robot's own software inherent handle the movement functionalities. This solution is cost effective compared to the purchase of KUKA.ForceTorqueControl package. Also this solution would be better than the cost-effective method mentioned in as it does not necessitate the installation of a alternate robot operating software, and it can make use of features offered by the robot's native software.This strategy aligns with the objectives of optimizing operational efficiency and reducing costs in robotic control systems, making it a valuable addition to the discourse in this thesis. The designing and implementation of the proposed system would be discussed in the upcoming chapters.

Chapter 3

Designing of Navigation system

The need for a robust hand-guidance system is paramount in applications requiring high precision and adaptability. The design of a new hand-guidance system for the Industrial robot is described in this chapter, with a focus on integrating current technologies like C3 bridge, python and KRL(Kuka Robot Language) for implementation. The highlight of this solution is that, it seamlessly integrates with the existing Kuka System Software(KSS), therby eliminating the need for the purchase of KUKA.ForceTorqueControl package. Also, utilizing the existing KSS, the system retains all the functionality and benefits of the OEM (Original Equipment Manufacturer)software.

3.1 System architecture

The proposed architecture for the navigation system is focused on integration of advance technologies which allows for the refinement of the robot movement based on the external forces and torque sensed during operation and controlled by a external system. The architecture consists of three critical components:

1. External Control System (Laptop)
2. Industrial Robot (KUKA robot)
3. Force/Torque Sensor

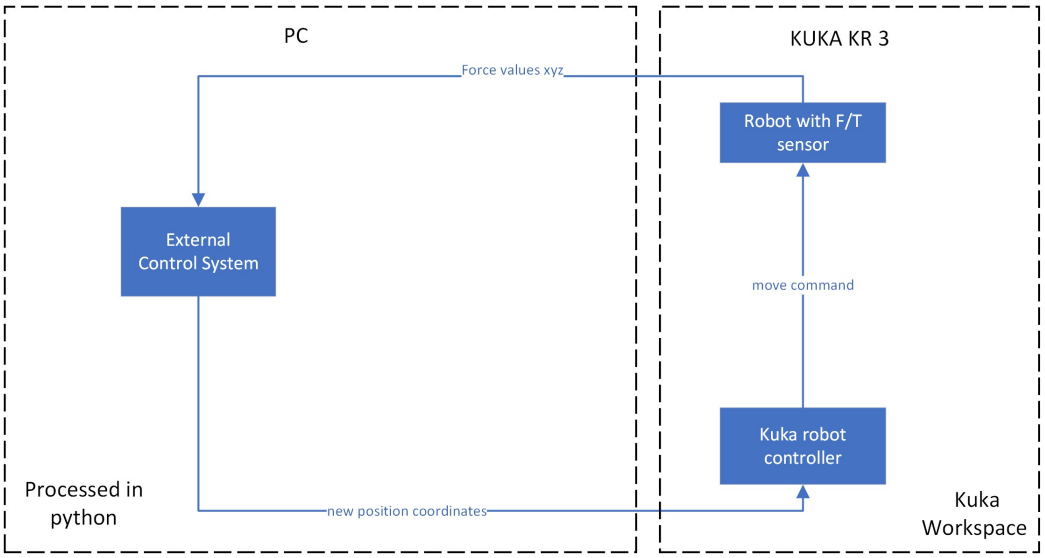

Figure 3.1: Block diagram

External Control System

In charge of gathering and analyzing sensory input, the External Control System is the center of the architecture. By directly interacting with the Kuka Robot and Force-Torque sensor, it functions as a central processing unit. The force values in the x, y, and z axes that are transmitted by the F/T sensor must be interpreted by this system in order to be converted into useful information. This converted value that is the distance to be moved is sent to the robot.

Industrial Robot

The Industrial Robot which in our case is the KUKA KR3 Agilus robot is equipped with F/T sensor which is attached to its end effector. The robot communication with the external control system and receives the continues updates of the distance to be moved which is then processed by the robot's proprietary programming language KRL and the robot moves.

F/T Sensor

The sensor is a crucial part of this system which detects multi-axial forces and torques that the robot encounters. The sensor's data output provides real-time feedback on the external force acting on the robot which is required for the navigation. This sensor data is transferred to the External Control system via UDP protocol.

3.2 System Workflow Diagram

The navigation system's design incorporates a User Interface(UI) displayed on the external Control system which has three buttons: Start, Stop and Close each representing different states of the system. The Navigation program starts working when the Start Button is pressed. The navigation program starts with the data collection from the F/T sensor. This sensor data is critical for determining the robot's movement coordinates or travel distance.

Upon acquiring the data, it is processed using certain algorithms to convert the force data acquired from sensor to a actionable coordinates or travel distance for the robot to move. As the data is received by the robot, it travels the distance obtained and continues to move as long as it receives updates from the external system. The external system keeps send the data as long as there is force applied on the sensor. This ongoing loop of feedback and action facilitates real-time adjustments to the robot's trajectory, ensuring precise navigation.

The process continues seamlessly until the Stop button is pressed. Only when the stop button is pressed, the navigation program halts and stops the transmission of data to the robot, effectively pausing its movement. To resume the navigation of the robot with hand guidance, pressing the Start buton again restarts the program and allows the robot to continue its movement.

The UI remains accessible throughout the process and exits only when the close button is pressed. This design is simple yet effective in management and operation within the navigation system, providing clear user control and robust system responsiveness.

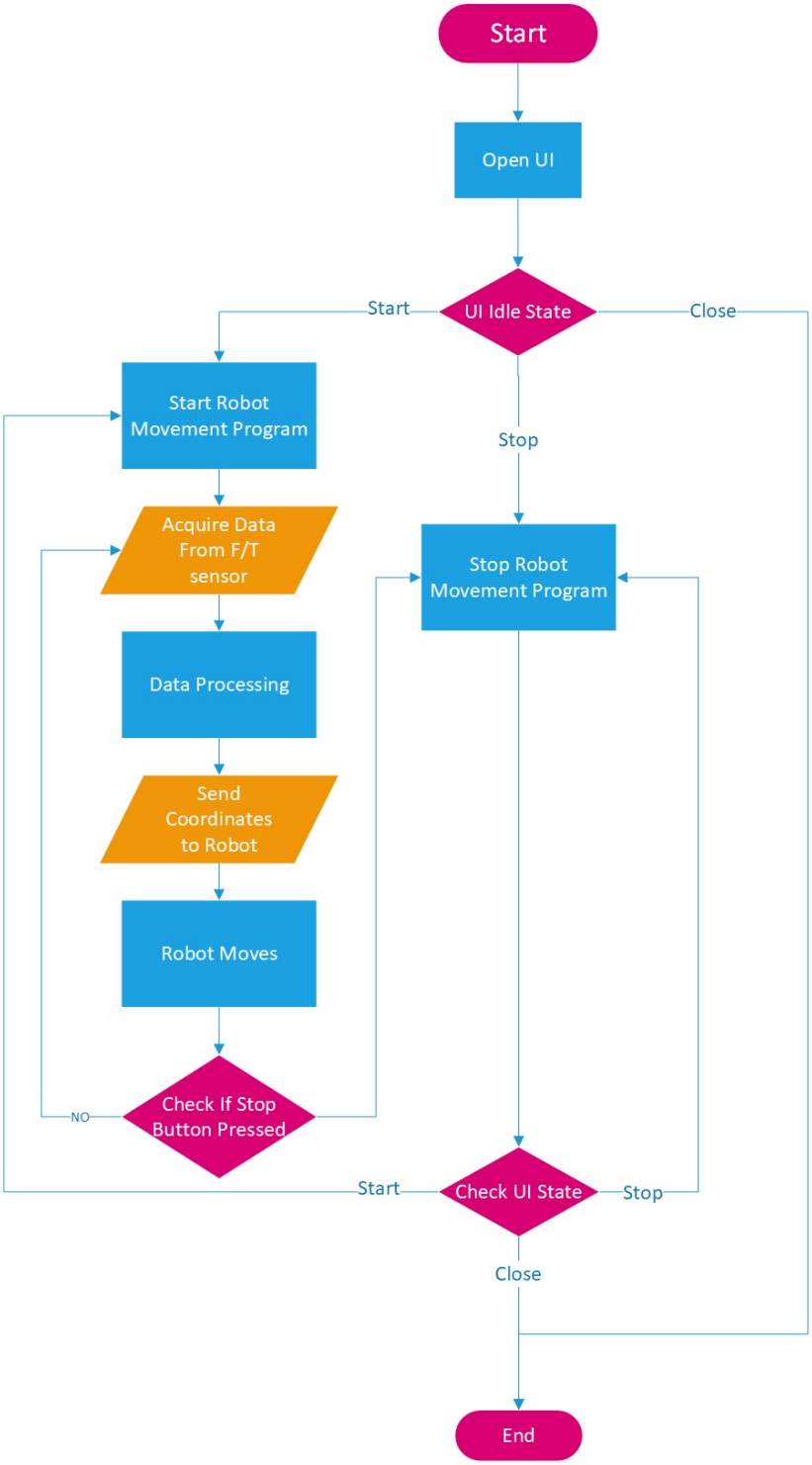

Figure 3.2: System Workflow Diagram

3.3 Hardware Components

The KUKA KR3 Agilus robot and the Schunk Force Torque Sensor are essential hardware components required to execute our suggested approach and provide accurate operational feedback. The KUKA KR C4 controller facilitates control and command functions. It is enhanced with a SmartPad HMI, which improves the user interface. In addition, a laptop or personal computer is needed for programming and computing duties. Ethernet cables are used to maintain connectivity between these components. Python is needed for scripting features, KRL (KUKA Robot Language) is needed for direct robotic programming, and C3Bridge is needed to connect different control systems and architectures.

3.3.1 Robot

The Industrial robot used for designing this navigation system is KUKA KR3 AGILUS robot. Its an lightweight agile robot. The newest robot in the KR AGILUS small robot series allows automation in cells as small as 600 x 600 mm. But because it is even smaller than the larger models in the KR AGILUS series, it is the perfect answer for small cell concepts like those needed in the 3C market (computers, communications, and consumer electronics). The KR 3 AGILUS is inexpensive, low maintenance, and extremely dependable because of its clever design. It is perfect for satisfying the demands of many industries, especially the electronics sector, which is one of the biggest and fastest-growing areas for automation with a payload capacity of 3 kg and a reach of 540 mm. In their payload categories, the KR AGILUS robots' performance is unparalleled. The robots also include an integrated energy supply system, high speed, short cycle duration, and six axes. They are capable of carrying out odd jobs in any installation configuration. All KR AGILUS models are operated with the service-proven KR C4 compact or KR C4 smallsize-2, the universal control technology for all KUKA robot models. The KR 3 AGILUS is very dependable, reasonably priced, and requires little upkeep. Reliable technology, sturdy components, and proven KUKA quality offer the best output and highest availability, which translates into a maximum return on investment and a low total cost of ownership [12].

Figure 3.3: Kuka KR3 Agilus [12]

3.3.2 Kuka KR C4 controller

Pioneering both today's and tomorrow's automation is the KR C4 controller. It lowers integration, upkeep, and service expenses. In addition, the systems' long-term flexibility and efficiency are raised because of open, shared industry standards. The KR C4 software architecture combines safety control, motion control (like KUKA.ForceTorqueControl), PLC control, and robot control. Infrastructure and a database are shared by all controllers. As a result, automation is now and in the future both more powerful and simpler[13].

Figure 3.4: Kuka KR C4 controller[14]

3.3.3 Smartpad HMI

The KUKA smartPAD teach pendant was made to make even the most difficult operating duties simple to learn. Even for inexperienced users, it is simple to use and may be distributed globally. All KUKA robots running on KSS and Sunrise.OS can be operated in the desired language using the KUKA smartPAD. The KUKA smartPAD is hot-pluggable and can thus be connected and disconnected at any time. The context-sensitive interface of the KUKA smartPAD only displays the options relevant at the moment of operation. Thanks to the intuitive operator guidance, less experienced and expert users alike can work quickly and efficiently with a minimum of training[15].

Figure 3.5: KUKA Smartpad HMI[14]

3.3.4 Force Torque Sensor

The Schunk force/torque sensor offers a highly efficient and precise solution for industrial automation needs. This sensor's small form size and integrated electronics maximize space utilization and simplify the setup procedure. With its two measurement ranges accessible through a web-based interface, it is particularly useful for scenarios requiring flexible calibration capabilities. This allows for adaptation to a wide range of operating needs[16].

Figure 3.6: Force Torque Sensor[16]

Via a direct plug-and-work software module, the sensor guarantees compatibility with robotic systems like KUKA and Universal Robots, enabling smooth integration into current infrastructures. Because of improved production techniques, the sensor is still economically feasible even with its high level of precision. Its sturdy design supports a high overload capacity and resilience against damage from temporary overloads means that it is made to last.

Improved user interaction is provided via an on-board LED display, which provides instantaneous visual feedback on the state of the sensor without requiring direct interface with the control system.The Ethernet Axia sensor consists 3 LED for Link/Activity (L/A), Diagnostic (DIAG), and Status. These LED has 3 colors- off, red, green[17].

LED Color	**Status**	**Description**
Off	No power or connection activity	Connection/activity is not detected.
Green	Link/Activity	Remains green for five seconds after every connection activity.

Table 3.1: EtherNET L/A LED[17]

LED Color	Status	Description
Off	No power supply	The sensor is not supplied with power.
Green	Normal operation	The electronics of the sensor are working and can communicate.
Yellow	Detection range exceeded	The forces and torques applied to the sensor exceed the permitted ranges. Reduce load or use larger calibration.
Red (flashing at 1 Hz)	Calibration error	The sensor does not refer to a calibration range or has a checksum error.
Red (flashing at 10 Hz)	Communication error	The sensor is not able to transmit data via the communication protocol.
Red	Status code error	Information about the error record.

Table 3.2: Sensor status LED[17]

LED color	Status	Description
Flashes green	Before operation	Defined by the communication/protocol standard.
Green	Ready for operation	No errors were found.
Red	Error	Indicates an error reported by the internal electronic components. In addition, the LED remains red for five seconds after a UART error.

Table 3.3: Diag LED[17]

Strong connectivity is provided by a very flexible shielded cable with an M8 or M12 connector. This cable guarantees consistent power supply and signal transmission while protecting against mechanical and electrical interferences. Additionally, the sensor offers a flexible and dependable performance in challenging industrial environments by supporting Ethernet or EtherCAT connections on its control line.

The Ethernet Axia sensor offers the following features:

- Calibrated force/torque data
- Bias functionality
- Programmable low-pass filtering with cut-off frequency
- Tool transformation
- Thresholding

- LED indicator for Link/Activity (L/A), Diagnostic (DIAG), and Status
- Compatible with the ATI Net F/T sensor UDP interface and Java demo application
- Compatible with parts of the ATI Net F/T web interface

UDP Interface Using RDT

The Axia Ethernet system is capable of transmitting data at a maximum frequency of 7912 Hz via UDP over Ethernet. This rapid data transmission technique is known as Raw Data Transfer (RDT). An RDT packet includes various types of data such as forces, torques, and status codes from the Ethernet Axia.

Command Code	**Command**	**Purpose**	**Response**
0x0000	Stop	Stop sending RDT packets over UDP.	None
0x0001	Start single block	Start sending RDT packets over UDP to the requestor, single blocks only, regardless of the RDT buffer size setting. Use the Count field to send a specific number of packets, 0 = unlimited.	RDT record(s).
0x0002	Start multiblock	Start sending RDT packets over UDP to the requestor, how many RDT packets are blocked depends on the RDT buffer size setting. Use the Count field to send a specific number of packets, 0 = unlimited.	RDT record(s).
0x0042	Bias	Set Software Bias.	None

Table 3.4: RDT Commands[17]

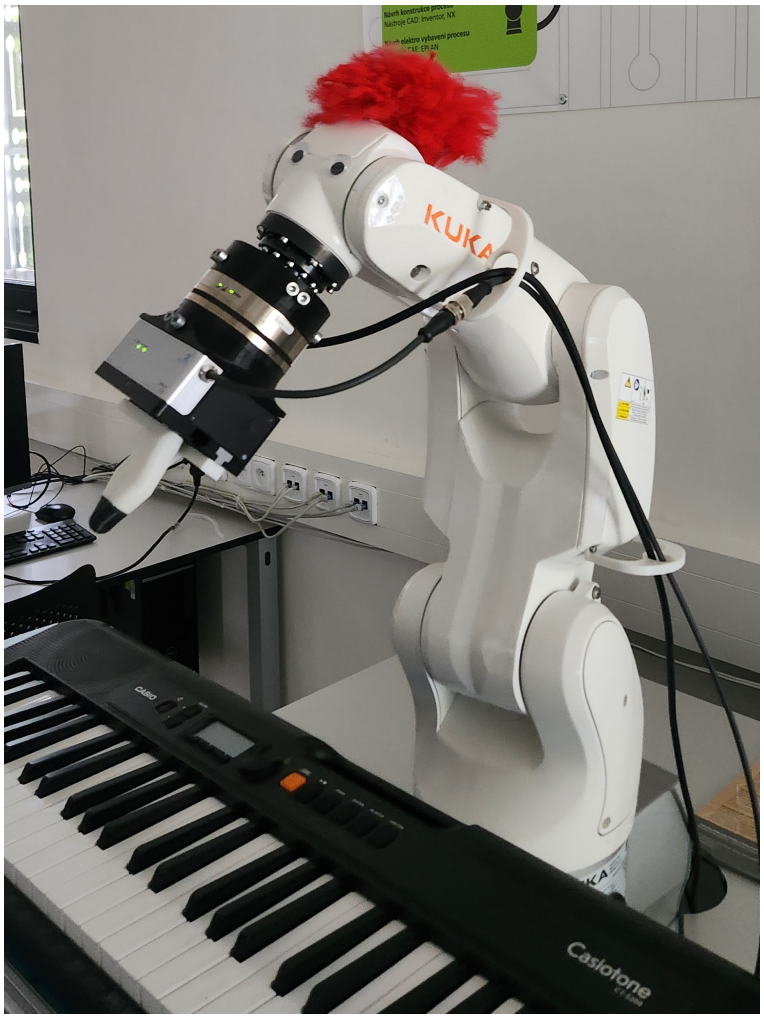

Figure 3.7: Sensor fixed to the robot

3.4 Software Part

This section describes the software tools and platforms used to develop and implement the hand-guidance navigation system, emphasizing their configuration, functionality, and the roles they play in the overall system.

3.4.1 Python

Python is used because of its powerful libraries and frameworks, which effectively enable complicated data processing and system control tasks. Its broad ecosystem and its simplicity in integrating with different hardware interfaces make it the perfect option for our project.

Figure 3.8: Python logo[18]

In our solution, python is the most critical part. The entire control algorithm is written in python which is used in all aspects of our system. Major tasks such as acquiring data from force torque sensor, communication with the robot, making real-time calculations are performed in python. The language's versatility allows the development of a responsive and flexible control system that can adapt to the dynamic requirements of hand-guidance navigation.

3.4.2 C3 Bridge

The C3 Bridge acts as a middleware that facilitates communication between high-level sofware application and low-level robot control mechanisms. It is useful for translating the commands from Laptop to a format that the robot's control unit can understand

As control algorithm is developed using python in Laptop. So for the integration of the control algorithm with the robot, C3 bridge is essential. In our implementation, the robot variables needs to be updated using which the robot moves and the variables are updated using the commands and format provided by the C3 bridge which translated it to a robot understandable format.

Figure 3.9: C3 Bridge logo

3.4.3 KUKA WorkVisual

KUKA WorkVisual is a software package that is used for configuring, programming and visualizing KUKA robots and their components. It provides a unified environment for all stages of development from hardware configuration to programming and maintenance.

WorkVisual is used in our project to configure the robot and develop KRL program. The movement program written in KRL is responsible for using the updated variable from external control system to move the robo and WorkVisual is used for writing that program. It allows for visualization of robot's workspace, setting up initial parameters etc.

3.4.4 Operating System

The operating systems involved in our project include Windows on the laptop and KUKA System Software(KSS) on the robot controller.

- Windows: The external control system that is the Laptop used in our implementation of the navigation system runs on Windows OS. It provides a familiar and versatile environment for developing the control software by offering broad support for development tools like python, KUKA WorkVisual.
- KSS(KUKA System Software): KSS is the KUKA robot's operating system, which is designed specifically for robot management and operation. It handles all aspects of robot behavior including motion control, safety and communication with peripheral devices.

Chapter 4

Implementation of Navigation System

4.1 Configuration of System Components

Before implementing this solution we need to have certain configurations that needs to done in the robot and sensor side.

4.1.1 Robot setup

The robot needs to be connected to the Laptop via Ethernet. To enable the communication between Laptop and Robot via Ethernet, the robot needs to have C3 Bridge installed. Once it's done, the robot's network configuration needs to be noted. The network configuration in Laptop should be configured to connect with the robot. Modify the Internet Protocol Version 4(IPv4) settings of the Ethernet Properties to match the IP address and DNS of the robot.

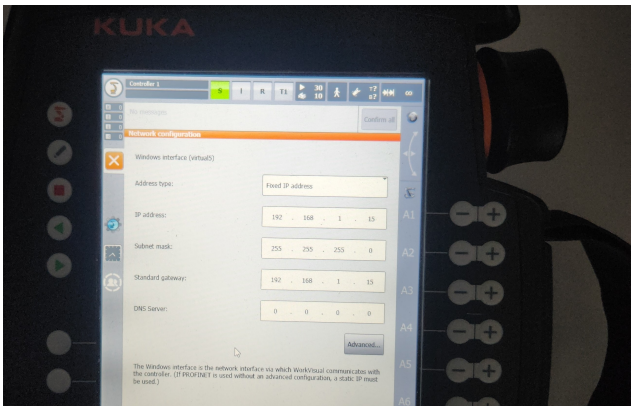

Figure 4.1: Robot's Network Configuration

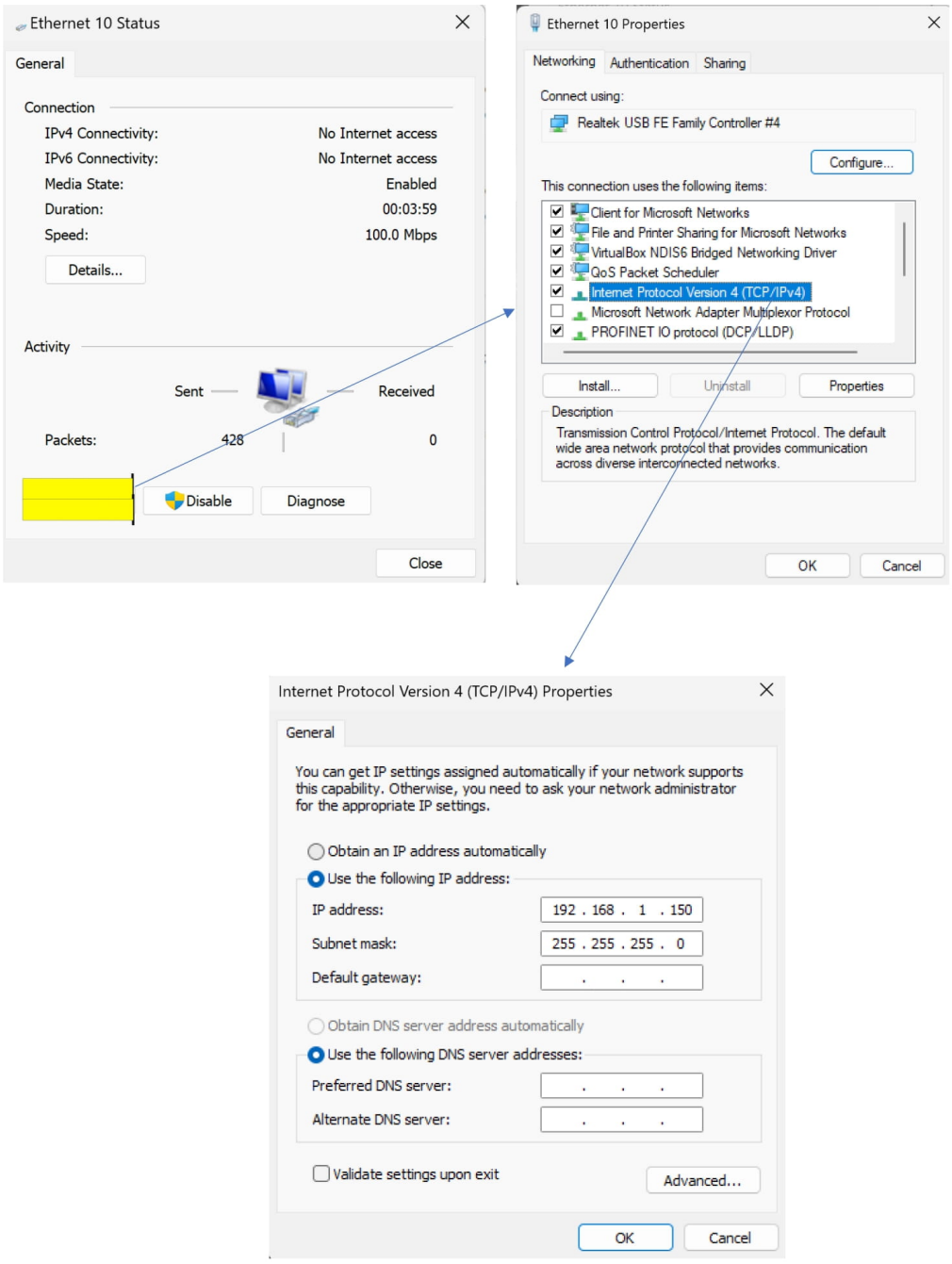

Figure 4.2: Network Configuration in Laptop

4.1.2 Sensor Setup

The sensor is to be mounted on the robot's end effector. The sensor is equipped with two cables: one for power supply and the other for communication via an Ethernet connection. In our proposed system, the sensor's communication cable is linked to a laptop. It is imperative to adjust the Internet Protocol Version 4 (IPv4) settings within the sensor's Ethernet properties. Specifically, the IP address and DNS settings must be configured. These parameters should be recorded for integration into the control algorithm, facilitating communication with the sensor.

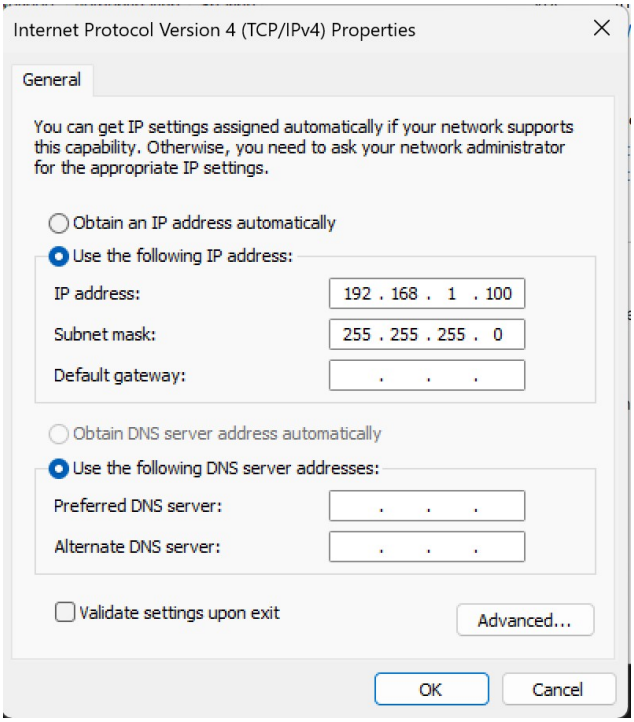

Figure 4.3: Network Configuration of Sensor in Laptop

4.2 Software development

The system operates across dual platforms. The robot's movements are controlled through a program written in KUKA Robot Language (KRL). This movement program receives updates on the required coordinates from a laptop, which processes the incoming sensor data into actionable movement coordinates for the robot.

4.2.1 Control Algorithm in Laptop

Upon initiation of the program, the user interface (UI) is activated. The control sequence begins when the 'Start' button within the UI is engaged. This action establishes connectivity between the

laptop and both the robot and the sensor. Subsequent to the establishment of these connections, a loop starts and the sensor data is captured using the UDP (User Datagram Protocol).

Figure 4.4: External Control System(Laptop) with UI

Sensor Package

A package with a Python class "FT_SENSOR" was developed for establishing and managing the communication of sensor with the laptop. This communication is executed with the combination of HTTP requests and UDP streaming, leveraging the sensor's API to fetch and stream data efficiently.

The sensor's configurations and parameters required for operation are fetched via HTTP requests to the sensor's API endpoint('netftapi2.xml'). The python class initializes by configuring a UDP socket and binds it to an available port on the laptop, setting the groundwork for data streaming. The parameters that are obtained include raw force data, raw torque data, conversion factor for the raw sensor outputs, bias settings, maximum range of sensor readings. The major functions within the class handle specific communication aspects:

The 'read_device_settings' method fetches current settings of the sensor by parsing the XML response.

Code of the read_data_settings method

.

```
    def read_device_settings(self):
        soup=self._read_netftapi2()
        device_status = int(soup.find('runstat').text,16)
```

```
        if soup.find('scfgfu').text != 'N':
            raise Exception('ATI Net F/T must use MKS units')
        if soup.find('scfgtu').text != 'Nm':
            raise Exception('ATI Net F/T must use MKS units')
        if soup.find('comrdte').text != "Enabled":
            raise Exception('ATI Net F/T must have RDT enabled')

        cfgcpf=float(soup.find('cfgcpf').text)
        cfgcpt=float(soup.find('cfgcpt').text)

        def _to_array(s):
            return np.fromstring(soup.find(s).text, dtype=np.float64, sep=';' )

        conv=np.asarray([cfgcpt, cfgcpt, cfgcpt, cfgcpf, cfgcpf, cfgcpf], dtype=np.float64)
        maxrange=_to_array('cfgmr')
        bias=np.divide(_to_array('setbias'), conv)
        ft1=_to_array('runft')
        ft=np.divide(np.append(ft1[3:6],ft1[0:3]), conv)
        ipaddress=soup.find('netip').text
        rdt_rate=int(soup.find('comrdtrate').text)

    return NET_FT_device_settings(ft, conv, maxrange, bias, ipaddress, rdt_rate, device_status)
```

The 'set_tare_from_ft' and 'clear_tare' methods manage the sensor's tare adjustment, which is crucial for resetting the sensor's zero point to account for any pre-existing forces or biases. The 'start_streaming' and 'stop_streaming' methods are the most critical methods which manage the starting and stopping of the the data streaming via UDP.

Code

```
    def start_streaming(self):
        sample_count=10*self.device_settings.rdt_rate
        dat=struct.pack('>HHI',0x1234, 0x0002, sample_count)
        self.socket.sendto(dat, (self.host, 49152))
        self._streaming=True
        self._last_streaming_command_time=time.time()

    def stop_streaming(self):
        dat=struct.pack('>HHI',0x1234, 0x0000, 0)
```

```
        self.socket.sendto(dat, (self.host, 49152))
        self._streaming=False
        self._last_streaming_command_time=time.time()
```

The try_read_ft_streaming method in Python manages data streaming from a networked device, specifically checking and maintaining a live data stream. It listens for incoming data packets on a UDP socket using a non-blocking select.select call. If a packet is received, it is unpacked and processed to extract and scale force and torque measurements, adjusting for calibration. If no valid data is received after multiple attempts or exceptions, it returns an indication of failure.

Code for reading sensor data

```
    def try_read_ft_streaming(self, timeout=0):
        #Re-up the streaming if running out of packets
        if (time.time() - self._last_streaming_command_time) > 5:
            if (self._streaming):
                self.start_streaming()

        s=self.socket
        s_list=[s]
        buf=None

        timeout1=timeout
        drop_count=0
        while(True):
            res=select.select(s_list, [], s_list, timeout1)
            if len(res[0]) == 0 and len(res[2])==0:
                break
            try:
                (buf, addr)=s.recvfrom(1024)
            except:
                return False, None, 0

            if (drop_count > 100):
                break
            timeout1=0
            drop_count+=1
        if (buf is None):
            return False, None, 0
```

```
    rdt_sequence, ft_sequence, status, Fx, Fy, Fz, Tx, Ty, Tz \
        =struct.unpack('>IIIiiiiii', buf)
    ft=np.divide(np.asarray([Tx, Ty, Tz, Fx, Fy, Fz]), \
        self.device_settings.conv)-self.tare

    return True, ft, status, rdt_sequence
```

A specialized sensor package is employed for communication with sensor, which can tare the sensor value, pack and unpack the data, etc. From this unpackaged data, the essential components, namely the forces along the x, y, and z axes (Fx, Fy, Fz), are extracted. To mitigate potential disruptions or errors resulting from packet loss, a method of averaging the data packets is implemented. Importantly, the quantity of packets averaged is configurable, allowing for adaptive data fidelity.

Code for data packets averaging:

```
packet_count = Parameters.packet_group# packets to be averaged for force calculation
        accumulated_fx = accumulated_fy = accumulated_fz = 0.0
        sensor.start_streaming()

        for _ in range(packet_count):

            success, ft_streaming, sequence = sensor.read_ft_streaming(timeout=1)
            if success:
                _, _, _, fx, fy, fz = ft_streaming
                accumulated_fx += fx
                accumulated_fy += fy
                accumulated_fz += fz
            else:
                logger.info("No data Received")

        if packet_count > 0:       #
            avg_fx = accumulated_fx / packet_count
            avg_fy = accumulated_fy / packet_count
            avg_fz = accumulated_fz / packet_count

        else:
            logger.error("set packet count value greater than zero")
```

```
avg_fx = avg_fy = avg_fz = 0.0
```

Subsequent to data averaging, the resultant metrics that is the average force along x (avg fx), y (avg fy), and z (avg fz) are computed. To differentiate between incidental contact and deliberate force intended to direct the robot, a modifiable force threshold is established. Crossing this threshold with any of the force readings (X, Y, or Z) triggers the conversion of these force values into relative distances for robot movement.

Code for force threshold:

```
# Determine if movement is needed based on force threshold
        if avg_fx>force_threshold or avg_fx<(-1*force_threshold):
            x_force = True
        else:
            x_force=False

        if avg_fy>force_threshold or avg_fy<(-1*force_threshold):
            y_force = True
        else:
            y_force=False

        if avg_fz>force_threshold or avg_fz<(-1*force_threshold):
            z_force = True
        else:
            z_force=False
        if x_force or y_force or z_force:
        #robot movement program here
        else:
            robot_stop_move()
            logger.info(f"threshold not crossed")
```

This conversion process directly maps the sensor's force values to corresponding robot axes. For instance, applying force along the positive x-axis of the sensor correlates to movement along the positive y-axis of the robot. These force values are further adjusted by an impedance factor, which calibrates the magnitude of the robot's step response to each unit of force applied.

Code for mapping sensor axis to robot axis:

```
if x_force or y_force or z_force:
```

```
logger.info(f"robot is moving")
x_move = impedance_factor*avg_fy
y_move = impedance_factor*avg_fx
z_move = -1*impedance_factor*avg_fz
```

Finally, the computed relative distances are relayed to the robot by updating a predefined variable, utilizing the write function from the KUKALIB package, which interfaces with the C3Bridge. This ensures that the robot moves according to the specified parameters, effectively responding to the dynamic inputs provided by the sensor data.

Code for updating variable in robot workspace:

```
rel_move_distance = np.array([x_move, y_move, z_move, 0.0, 0.0, 0.0], dtype="float")
            robot.client.write("C3BI_CONT", '{X ' + str(rel_move_distance[0]) +
                                ', Y ' + str(rel_move_distance[1]) +
                                ', Z ' + str(rel_move_distance[2]) + '}')
```

4.2.2 KRL Programming

KUKA Robot Language(KRL) is the primary programming language that is used for controlling KUKA robots. It is the proprietary language specifically tailored for robotic applications, which enables precise control of robot motions, operations and interactions with peripheral devices. The KRL code for this solution is designed to implement a basic continuous motion control for the KUKA robot based on the dynamic variable updates received from the external System.

The code contain various core aspects of KRL programming such as the motion commands, data handling and loop structures. The KRL program can be modified in KUKA WorkVisual Software or directly from the SmartPAD. In the KRL explorer there are various folders and for our solutions, we use two folders, Program and System. The KRL code is written in 'test_move.src' under Program folder and variables are declared in 'config.dat' file in system folder.

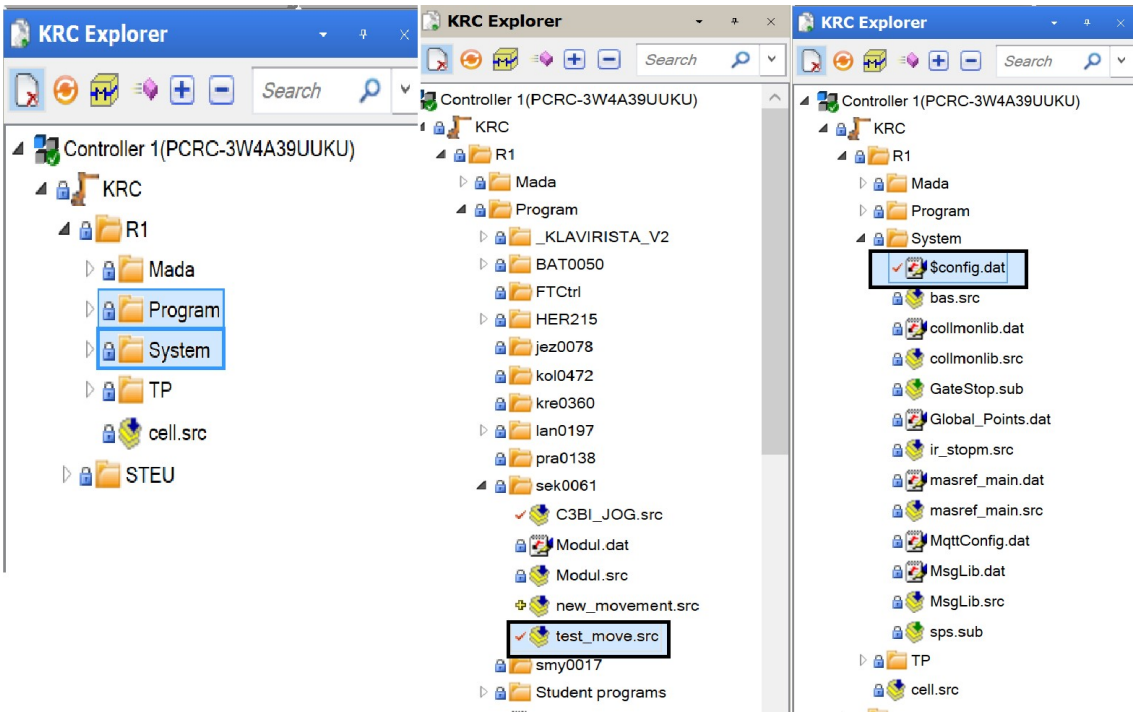

Figure 4.5: Navigating to the files in WorkVisual

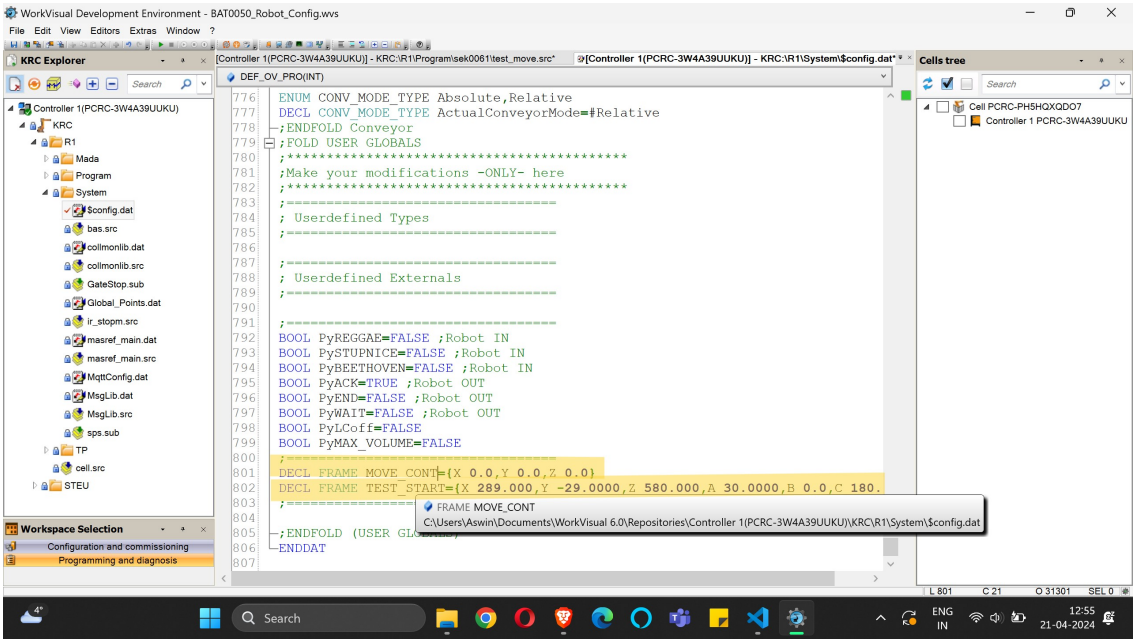

Figure 4.6: Config file with the variable required

Code for Initialization and Access Control:

```
&ACCESS RVO
DEF test_move()
```

These line specifies the level of access required to execute the program and defines a new function test_move which encapsulates the robotic operations

Code For Basic Configuration:

```
BAS(#INITMOV, 0)
PTP $AXIS_ACT
```

In this part of the code, the first line initializes the motion settings of the robot, setting them to a defined baseline configuration and the second line directs the robot to move the position defined by the variable which typically represents the current axis position.

Code For Environmental Setup:

```
$BASE = {X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0}
$TOOL = {X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0}
```

The base and tool coordinate frames of the robot to the origin are set in this part.

Code For Continuous Motion Implementation:

```
MOVE_CONT = {X 0.0, Y 0.0, Z 0.0}
PTP TEST_START
LOOP
    CONTINUE
    PTP_REL MOVE_CONT C_PTP
ENDLOOP
```

This part is important section of the code which handles the movement. First it initializes the movement vector with no displacement. Then the robot moves to the starting position designated by the variable 'TEST_START'. After it reaches the starting position, a loop starts executing the enclosed instruction which is to move the robot based on relative point-to-point motion command, where the variable 'MOVE_CONT' is continuously updated from the laptop based on the force applied on the sensor. The 'C_PTP' configuration constant enables the continues point to point movement so that the robot movement looks smooth rather than stopping at each point it reaches.

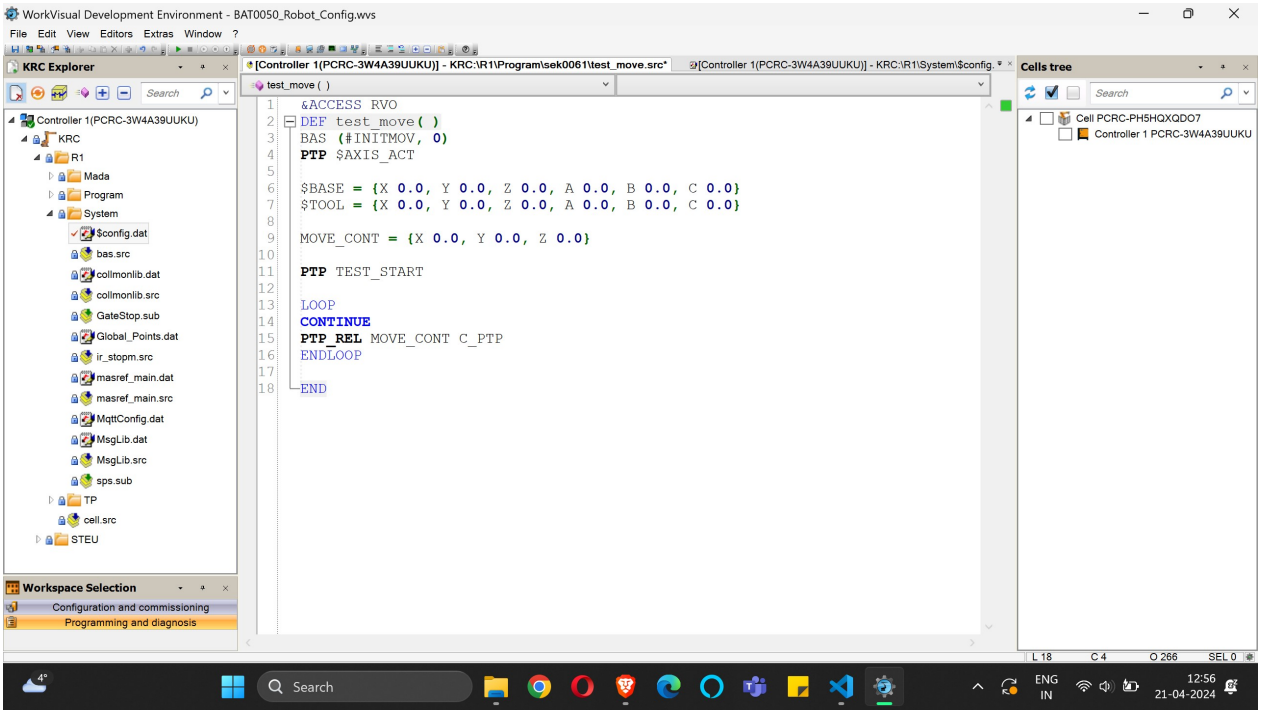

Figure 4.7: KRL code view in KUKA Work Visual

Figure 4.8: KRL code view in KUKA SmartPad HMI

The entire code demonstrates a practical application of networked robotics, where the robot can perform tasks based on the real-time data inputs which enhances its autonomy and flexibility in industrial settings. In this case, this code needs to run in automatic mode first and once the program in the Laptop is running, the robot starts moving based on the interaction in UI and force applied for the movement.

Another critical part is that, the robot needs to be set in automatic mode before running the program. the KRL program needs to be run in the automatic mode, so that the loop runs. The figure 4.9 highlights the part for changing to automatic mode. The key which is shown using yellow arrow should be turned to settings mode and the red arrow shows which mode it needs to be set for the program to be run.

Figure 4.9: Robot modes

4.3 User-Interface Development

The Design and implementation of the UI for our robot navigation system is a simple yet effective structure. The three main control buttons on the user interface (UI), which was created with the help of the Tkinter toolkit in Python, are Start, Stop, and Close. Each of these buttons has a specific purpose in controlling how the robot operates.

The "Robot Navigation System" interface is designed with a simple look that puts an emphasis on clarity and use. Fullscreen mode is an optional feature of the UI main window that improves visibility in industrial settings. The control buttons are arranged in a specific frame using a grid pattern to guarantee precise positioning. Using the Tkinter ttk, each button has been styled with considerable padding and an enlarged font for ease of Use.

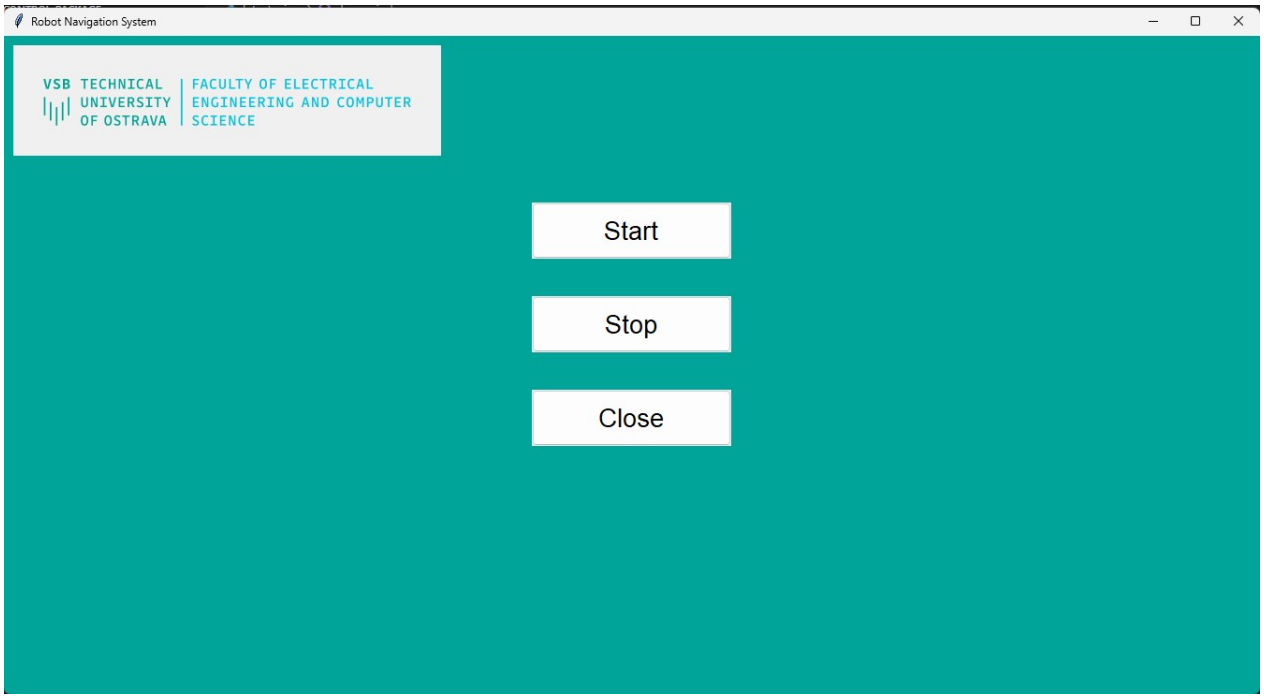

Figure 4.10: UI

4.3.1 Control Elements

There are three buttons present in the UI

Start Button

This button the initiates the robot movement program. When this is pressed, the data acquisition from sensor, then data processing to convert force data to the relative distance and writing it to the robot variable happens in a loop.

Stop Button

This button stops the robot movement as it stops the data acquisition,data processing and writing functions and updates the final relative distance to 0, so that the robot stops and not continues moving

Close Button

This button's purpose is to exit the UI. For operational safety, this button triggers a verification process for checking if stop button has been pressed before closing the UI, cautioning to stop any active robot movements. Once its done, the UI is closed.

4.3.2 Multithreading Implementation

In the architecture of our user interface (UI) for controlling robotic operations, concurrency is utilized to enhance responsiveness and efficiency. The system is structured to operate with two principal threads that separate UI management from backend processes.

UI Thread

This thread is dedicated solely to the UI, whose primary function is to monitor and respond to button presses. The separation of the UI into its own thread is implemented to prevent the graphical interface becoming unresponsive when heavy processing tasks occur. This allows for a smooth and responsive UI and user experience, which allows seamless start, stop, close operations

Data Handling Thread

Concurrently, another thread is tasked with the critical functions of data acquisition, processing and writing. This handles all the interactions with the sensor , robot and all the computational tasks as needed. By isolating these data-intensive operations from UI thread, the program can run efficiently in the background without interfering with user's ability to control the robot.

Chapter 5

Testing and Validation

5.1 Tests done

Functional testing of the Industrial robot navigation system was performed to verify that the system functions correctly to its defined specifications and requirements. The tests performed and their outcomes are mentioned below.

5.1.1 System Initialization test

The purpose of this test is to ensure that the robotic system and all components initialize correctly and are ready for operation.

Power up Procedures:

1. Action:Power on all the components such as KUKA robot, F/T sensor and the control computer.
2. Expected Outcome: All the devices were on without any errors. The indicator lights in sensor, KUKA Smartpad should confirm that the hardware components are active.
3. Observed Outcome: The indicator lights in the sensor and KUKA Smartpad confirmed that the hardware components are active without any errors.

Figure 5.1: Robot Start-Up Success

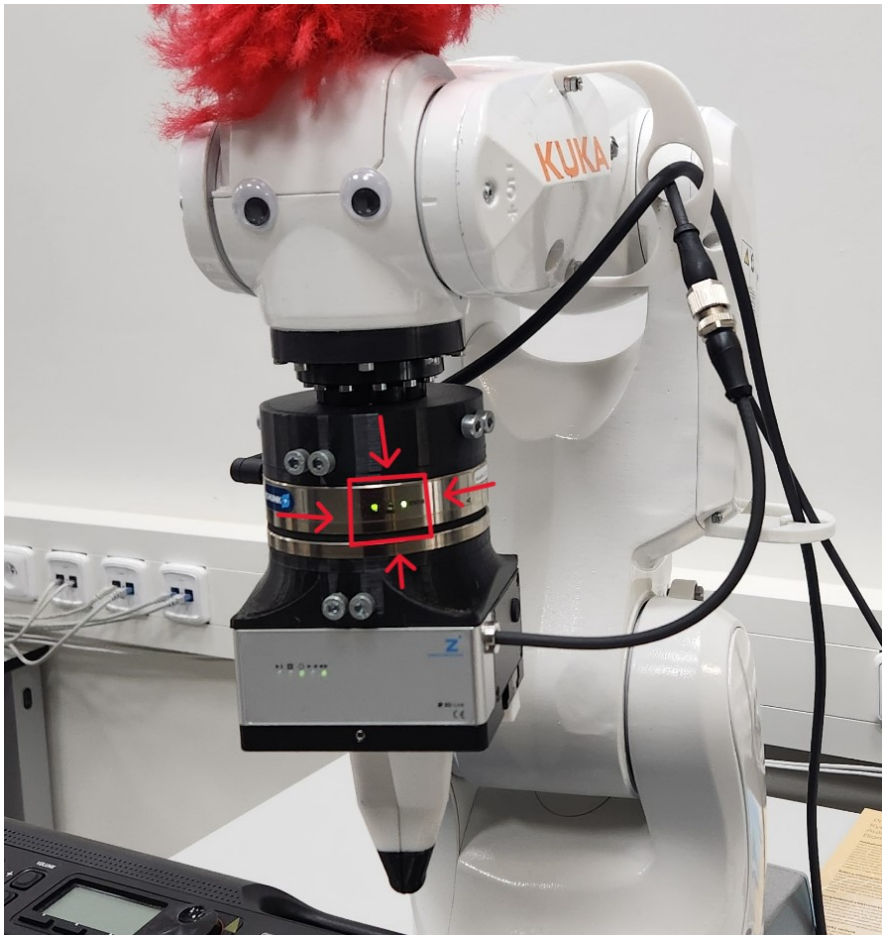

Figure 5.2: Sensor LED indicator showing start-up success

Communication Test

1. Action: Launch the python scripts used for communication and ensure connection of the laptop with the robot and sensor is established.
2. Expected Outcome: Connection success message for both the robot and sensor should be printed in the Output terminal. Variable updating command in python should change the value in robot Workspace.
3. Observed Outcome: The scripts ran as expected and the Connection success messages were printed. Also, the variable was updated in the robot workspace from python.

5.1.2 Movement Execution test

This test ensures that the robot moves to the exact position

1. Action: Sending the command to the robot to update the variable 'TEST_MOVE' with the values I send and the robot moves to that position
2. Expected Outcome: Checking the coordinates which is updated to the variable is the final position of the robot(noted from Smartpad) after it moves based on the variable updated
3. Observed Outcome: The robot moves perfectly to the coordinates which was updated to the variable 'TEST_MOVE'.

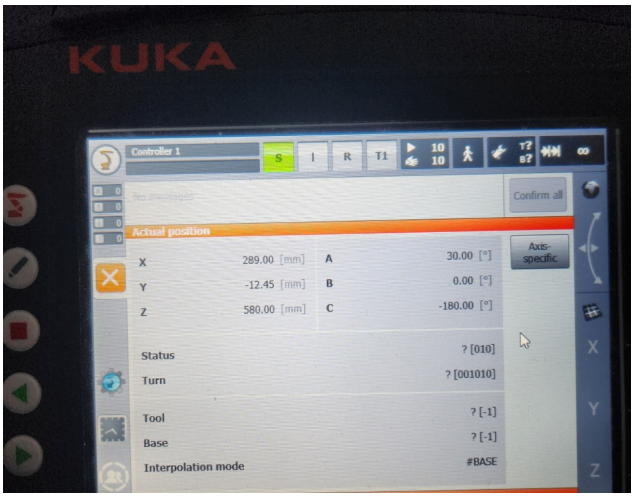

Figure 5.3: Robot position displaying in Smartpad

5.1.3 Threshold filtering test

This tests ensure if the threshold filter work correctly and ignores the accidental touches or slight disturbances.

1. Action: Run the program, set different threshold values and then apply forces below and above the threshold.

2. Expected Outcome: The Robot moves after it crosses the threshold and doesn't move for the value less than the threshold

3. Observed Outcome: The system behaved as expected without any errors.

5.1.4 Parameters adjustment test

This test determine how changes in system parameters such as robot speed, impedance factor, number of packets average, force threshold affect the performance and behaviour of the robotic navigation system. This helps in optimizing the performance of the robotic navigation system.

Robot Speed Adjustment

- Objective: The primary objective of adjusting the speed of the robot was to evaluate the impact on the system's reaction time and stopping precision.

- Observations: It was observed that increasing robot's speed increases the system's reaction speed. This also enhanced the stopping precision, indication the system can handle high speeds without compromising on the accuracy.

Packet Count Adjustment

- Objective: The objective of this adjustment was to evaluate the influence on the smoothness and accuracy of robot's movements.

- Observations:It was observed that increasing the packet count which affects the number of averaging packets was found to decrease the movement accuracy. This indicates a trade-off between data smoothness and responsiveness of the navigation system. Higher packet count averaging may cause delay in processing which might cause this issue as movements are based on real-time sensor inputs.

Impedance Factor Adjustment

- Objective: The objective of this adjustment is to evaluate the influence on the robot's movement precision and speed.

- Observations: Decreasing the impedance factor resulted in greater movement precision and accuracy but reduced the robot's speed.

Force Threshold Adjustment

- Objective: The objective of this adjustment is to evaluate the influence on the robot's movement.
- Observations: Increasing the Force threshold makes the required force for the robot to move increases. So to navigate the robot, more force is required, when the threshold is low, even slight touch moves the robot

Parameter Adjusted	**Adjustment Description**	**Observed Effect**
Robot Speed	Increased speed	Increased reaction speed and improved stop precision
Packet Count	Higher number of packets averaged	Decreased movement accuracy
Impedance Factor	Decreased factor	Increased movement precision and accuracy, decreased speed
Force Threshold	Increased threshold	Requires greater force to initiate movement

Table 5.1: Effects of Parameter Adjustments on Robotic Performance

5.2 Limitations

One of the inherent Limitations in our system is due to the usage of Python as primary programming language because it has Global Interpreter Lock(GIL) which restrict the execution of multiple threads in parallel. This affects in scenario where multi-threads would enhance the performance by parallelizing data processing tasks. To mitigate the effects of GIL on our system, we can use multi-processing instead of multi-threading, thereby bypassing GIL and enhancing the processing capabilities. Also, optimizing the data processing algorithm can reduce required computational resource and can achieve high efficiency within the single-threaded limitation imposed b the GIL.

Another such issue if the communication protocol which we used in our solution that is UDP which is known for packet loss. In this current solution, averaging of data packets was done to ignore packet loss. Similarly different algorithm can be implemented to handle packet loss based on the requirement.

Chapter 6

Conclusion

This thesis presented the design, implementation, and testing of a novel hand-guidance system for industrial robots using a torque sensor, with the primary objective of developing a cost-effective and accessible alternative to existing systems like KUKA's ForceTorqueControl. The system utilized open-source software and standard hardware, and was successfully designed to integrate a standard force/torque sensor with a KUKA robot through a custom-developed software interface using Python and KRL. The emphasis was on simplicity, usability, and efficiency. The implementation detailed the configuration of hardware components and the development of software to handle real-time data streaming and robot control, with the integration of the C3 bridge playing a crucial role in facilitating robust communication between the Laptop and the robot.

The testing of the system under various operational conditions showed the system's reliability and precision in navigating the robot arm using hand-guidance, with performance metrics confirming its suitability for industrial applications and noting improvements over some proprietary systems. This research contributes to the field of industrial robotics by offering a viable alternative to proprietary hand-guidance systems, potentially lowering the cost and easy implementation for those looking to integrate advanced robotic technologies and providing insights into the integration of open-source software with industrial robotics. This expands customization and cost reduction possibilities and sets a framework for future developments in robotic hand-guidance systems, particularly in using standard components and non-proprietary software solutions. The developed system supports the operational needs of different settings and offers scalability and adaptability to different environments, reducing dependence on expensive proprietary solutions and fostering innovation and technological advancement in the field. Although the system meets many of the intended goals, several challenges were encountered, such as handling multiple communications at the same time, and it demands continuous improvements to manage unusual operational scenarios. Further research and development are recommended to enhance the interface, expand sensor capabilities, and conduct scalability tests to ensure the system can be adapted to larger or more complex robotic systems.

References

1. INTERNATIONAL FEDERATION OF ROBOTICS. *World Robotics 2023 Report*. September 2023. Available also from: `https://ifr.org/ifr-press-releases/news/world-robotics-2023-report-asia-ahead-of-europe-and-the-americas`.
2. GERAVAND, Milad; FLACCO, Fabrizio; DE LUCA, Alessandro. Human-robot physical interaction and collaboration using an industrial robot with a closed control architecture. In: *2013 IEEE international conference on robotics and Automation*. IEEE, 2013, pp. 4000–4007.
3. ABU-DAKKA, Fares J; NEMEC, Bojan; KRAMBERGER, Aljaž; BUCH, Anders Glent; KRÜGER, Norbert; UDE, Ales. Solving peg-in-hole tasks by human demonstration and exception strategies. *Industrial Robot: An International Journal*. 2014, vol. 41, no. 6, pp. 575–584.
4. GRUNINGER, Rolf; KUS, Elzbieta; HUPPI, Richard. Market study on adaptive robots for flexible manufacturing systems. In: *2009 IEEE International Conference on Mechatronics*. IEEE, 2009, pp. 1–7.
5. BASCETTA, Luca; FERRETTI, Gianni; MAGNANI, Gianantonio; ROCCO, Paolo. Walk-through programming for robotic manipulators based on admittance control. *Robotica*. 2013, vol. 31, no. 7, pp. 1143–1153.
6. KUHN, Stefan; GECKS, Thorsten; HENRICH, Dominik. Velocity control for safe robot guidance based on fused vision and force/torque data. In: *2006 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*. IEEE, 2006, pp. 485–492.
7. MASSA, Daniele; CALLEGARI, Massimo; CRISTALLI, Cristina. Manual guidance for industrial robot programming. *Industrial Robot: An International Journal*. 2015, vol. 42, no. 5, pp. 457–465.
8. NORBERTO PIRES, J; VEIGA, Germano; ARAÚJO, Ricardo. Programming-by-demonstration in the coworker scenario for SMEs. *Industrial Robot: An International Journal*. 2009, vol. 36, no. 1, pp. 73–83.
9. KUKA ROBOTER GMBH. *KUKA.ForceTorqueControl 3.0*. Augsburg, Germany: KUKA Roboter GmbH, 2013-02. Version: KST ForceTorqueControl 3.0 V2 en (PDF).

10. LOSKE, Jonas; BIESENBACH, Rolf. Force-torque sensor integration in industrial robot control. In: *15th International Workshop on Research and Education in Mechatronics (REM)*. 2014. Available from DOI: 10.1109/REM.2014.6920241.

11. GREGOR, Radovan; BABINEC, Andrej; DUCHOŇ, František; DOBIŠ, Michal. Hand guiding a virtual robot using a force sensor. *acta mechanica et automatica*. 2021, vol. 15, no. 3, pp. 177–186.

12. KUKA. *The New KR 3 AGILUS* [https://www.kuka.com/cs-cz/firma/tisk/novinky/2016/07/the-new-kr-3-agilus]. 2016-07.

13. KUKA. *KR C4: Compact Controller for Increased Efficiency and Flexibility*. 2023. Available also from: https://www.kuka.com/en-de/products/robot-systems/robot-controllers/kr%C2%A0c4.

14. THAI WELDING. *KUKA Robots: KR AGILUS*. 2023. Available also from: http://www.thaiwelding.com/th/kuka-robots/kr-agilus.

15. KUKA. *SmartPAD: Intuitive Robot Operation*. 2023. Available also from: https://www.kuka.com/en-au/products/robotics-systems/robot-controllers/smartpad. Accessed: 2023-04-28.

16. SCHUNK. *FT-AXIA Force/torque sensor*. 2024. Available also from: https://schunk.com/cz/cs/automatizace/silove/momentove-senzory/ft-axia/c/PGR_3907.

17. KG, SCHUNK GmbH & Co. *FTN-AXIA Commissioning Instructions*. Germany: SCHUNK GmbH & Co. KG, 2022-11. Version 05.00. Available: https://www.schunk.com.

18. PYTHON SOFTWARE FOUNDATION. *Python Community Logos*. 2023. Available also from: https://www.python.org/community/logos/.

Appendix A

Code

UI.py

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
from tkinter import PhotoImage
import move_relative



def close_window():
    move_relative.stop_script()
    if messagebox.askyesno("Close", "Are you sure you want to close?\n\nNote:
        Press stop before closing*"):
        root.quit()


# Set up the UI
root = tk.Tk()
root.title("Robot Navigation System")
# root.attributes('-fullscreen', True) # Set the window to full screen

# Load and display a logo in the corner of the UI
logo = PhotoImage(file="logo_2.png") # Ensure "logo_2.png" exists in your project
    directory
logo_label = tk.Label(root, image=logo)
```

```
logo_label.pack(anchor='nw', pady=10, padx=10)

# Create a frame for buttons to use grid layout
buttons_frame = tk.Frame(root)
buttons_frame.pack(pady=20)
buttons_frame.configure(background='#00A499')
# Buttons using grid within the buttons frame

style = ttk.Style()
style.configure('TButton', font=('Helvetica', 20), padding=(20, 10))

start_button = ttk.Button(buttons_frame, text="Start",command=move_relative.
    start_script,style='TButton')
start_button.grid(row=0, column=0, padx=10, pady=20)

stop_button = ttk.Button(buttons_frame, text="Stop",command=move_relative.
    stop_script, style='TButton')
stop_button.grid(row=1, column=0, padx=10, pady=20)

close_button = ttk.Button(buttons_frame, text="Close", command=close_window,style=
    'TButton')
close_button.grid(row=5, column=0, padx=10, pady=20)

# Configure background
root.configure(bg='#00A499')

# Ensure clean exit
root.protocol("WM_DELETE_WINDOW", close_window)

# Start the application
root.mainloop()
```

move_relative.py

```
from KUKALIB.kukaconnector import KUKA_Handler
from KUKALIB import Parameters as Parameters
import threading
import numpy as np
from sensor_connect import FT_CONNECT
import logging
import re
import time

# Setup basic logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Initializing global variables
running = threading.Event()
robot = KUKA_Handler(Parameters.ip_KUKACell, Parameters.port_KUKACell)
sensor = FT_CONNECT(net_ft_host=Parameters.ip_sensor)
script_thread = None

reset_position = Parameters.KUKA_reset_rel_move
force_threshold = Parameters.force_threshold
impedance_factor = Parameters.impedance_factor # adjust for step movement

def robot_stop_move():
    robot.client.write("MOVE_CONT", '{X ' + str(reset_position[0]) +
                       ', Y ' + str(reset_position[1]) +
                       ', Z ' + str(reset_position[2]) + '}')

def move_robot_wrist():
    # Open the connection to the robot
    robot.KUKA_Open()
    # tare sensor
    sensor.set_tare()

    running.set()
```

```
try:
    while running.is_set():
        packet_count = Parameters.packet_group# packets to be averaged for
            force calculation
        accumulated_fx = accumulated_fy = accumulated_fz = 0.0
        sensor.start_streaming()

        for _ in range(packet_count):

            success, ft_streaming, sequence = sensor.read_ft_streaming(timeout
                =1)
            if success:
                _, _, _, fx, fy, fz = ft_streaming
                accumulated_fx += fx
                accumulated_fy += fy
                accumulated_fz += fz
            else:
                logger.info("No data Received")

        if packet_count > 0:    #
            avg_fx = accumulated_fx / packet_count
            avg_fy = accumulated_fy / packet_count
            avg_fz = accumulated_fz / packet_count

        else:
            logger.error("set packet count value greater than zero")
            avg_fx = avg_fy = avg_fz = 0.0

        # Determine if movement is needed based on force threshold
        if avg_fx>force_threshold or avg_fx<(-1*force_threshold):
            x_force = True
        else:
            x_force=False

        if avg_fy>force_threshold or avg_fy<(-1*force_threshold):
            y_force = True
        else:
```

```
                y_force=False

            if avg_fz>force_threshold or avg_fz<(-1*force_threshold):
                z_force = True
            else:
                z_force=False

            if x_force or y_force or z_force:
                logger.info(f"robot is moving")
                x_move = impedance_factor*avg_fy
                y_move = impedance_factor*avg_fx
                z_move = -1*impedance_factor*avg_fz
                rel_move_distance = np.array([x_move, y_move, z_move, 0.0, 0.0,
                    0.0], dtype="float")
                robot.client.write("MOVE_CONT", '{X ' + str(rel_move_distance[0]) +
                                   ', Y ' + str(rel_move_distance[1]) +
                                   ', Z ' + str(rel_move_distance[2]) + '}')
            else:
                robot_stop_move()
                logger.info(f"threshold not crossed")

            sensor.stop_streaming()

    except KeyboardInterrupt:
        logger.info(f"Stopped by user ")
    finally:
        robot.KUKA_Close()
        sensor.stop_streaming()

def start_script():
    global script_thread

    if not running.is_set():
        running.set() # Signal that we intend to run
        script_thread = threading.Thread(target=move_robot_wrist)
        script_thread.start()

def stop_script():
```

```
    logger.info("its stopped")
    global robot
    robot.KUKA_Open()
    running.clear() # Signal to stop the loop in move_robot_wrist

    robot_stop_move()
    robot.KUKA_Close()
    sensor.stop_streaming()

if __name__ == '__main__':
    move_robot_wrist()
```

---

test_move.src

```
&ACCESS RVO
DEF test_move( )
BAS (#INITMOV, 0)
PTP $AXIS_ACT

$BASE = {X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0}
$TOOL = {X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0}

MOVE_CONT = {X 0.0, Y 0.0, Z 0.0}

PTP TEST_START

LOOP
CONTINUE
PTP_REL MOVE_CONT C_PTP
ENDLOOP

END
```