University of Wisconsin Milwaukee

UWM Digital Commons

Theses and Dissertations

December 2022

# An Ethercat Based Real-Time Control System Design for Wheelchair-Mounted 6dof Assistive Robotic Arm

Ivan Alexander Rulik Cote
*University of Wisconsin-Milwaukee*

Follow this and additional works at: https://dc.uwm.edu/etd

Part of the Artificial Intelligence and Robotics Commons

Recommended Citation
Rulik Cote, Ivan Alexander, "An Ethercat Based Real-Time Control System Design for Wheelchair-Mounted 6dof Assistive Robotic Arm" (2022). *Theses and Dissertations*. 3068.
https://dc.uwm.edu/etd/3068

This Thesis is brought to you for free and open access by UWM Digital Commons. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of UWM Digital Commons. For more information, please contact scholarlycommunicationteam-group@uwm.edu.

# AN ETHERCAT BASED REAL-TIME CONTROL SYSTEM DESIGN FOR WHEELCHAIR-MOUNTED 6DOF ASSISTIVE ROBOTIC ARM

by

Ivan Alexander Rulik Cote

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Engineering

at

The University of Wisconsin-Milwaukee

December 2022

ABSTRACT

# AN ETHERCAT BASED REAL-TIME CONTROL SYSTEM DESIGN FOR WHEELCHAIR-MOUNTED 6DOF ASSISTIVE ROBOTIC ARM

by

Ivan Alexander Rulik Cote

The University of Wisconsin-Milwaukee, 2022
Under the Supervision of Professor Mohammad Habibur Rahman

Numerous assistive robots for individuals with disabilities have been produced over the past ten years, but researchers have not completely exploited these robotic technologies to enable people with impairments to live independently, especially in respect to activities of daily living. (ADLs). For people with impairments, an assistive system can help them fulfill the requirements of typical ADLs. Assistive robots can help address future healthcare demands due to a growing need for caregivers, a scarcity of them, and an increase in the number of the elderly and people with disabilities. Enhancing functional independence while creating a superior human-machine interaction is one of the most important considerations in the design of these assistive technologies. Current solutions are not considering the new technologies like the EtherCAT field bus that brings better response times and modularity allowing to perform a broader range of ADLs. Thus, the objective of this research design a Real-Time Control Platform (RTCP) based on EtherCAT while integrating multimodal control method for robotic self-assistance that could help individuals with disabilities in performing self-care tasks daily. In this research, a control framework using joysticks seamlessly control a wheelchair and a wheelchair-mounted robotic arm. Custom circuitry was developed to connect the RTCP with the powered wheelchair. Multiple experiments were conducted to test the robotic system. The control method has been tested rigorously, maneuvering the robot at different velocities. The round-trip delay we set between the commands while controlling the assistive arm was 500 us. Tests performed showed that the proposed control system allowed individuals to perform some ADLs such as picking up and placing items with a completion time of less than 1 min for each task.

Key words: EtherCAT, real-time, assistive robot, 6DOF, multimodal control, wheelchair, motor dysfunction, activities of daily living

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ACKNOWLEDGEMENTS

I want to start by expressing my gratitude to my parents, Maria, and Karel (RIP), for setting a strong example for me and providing unwavering support throughout my studies and life in general. I owe a great deal of gratitude to Professor Rahman for his guidance and mentoring, which have helped me develop into a rounded engineer capable of taking on new challenges while continually learning and adapting. I am also grateful for the Biorobotics Laboratory, which has served as a second family to me while I have been living overseas. My colleagues have been very understanding of the challenges I have had, and I have done the same for them.

Finally, I want to thank my dear MJ for all that she has done to help me improve both personally and professionally. My never-ending inspiration pushing me to new heights. Despite the ups and downs of this life, I'm thrilled to share it with you.

# Chapter 1: Introduction

## 1.1 Importance of the Problem

Stroke, cerebral vascular accident, amyotrophic lateral sclerosis, spinal cord injury (SCI), trauma, and industrial injuries are all potential causes (McKee & Daneshvar, 2015). These accidents may lead the affected person to lose upper limb capability, limiting functional independence and preventing them from doing activities of daily living (ADLs) without external support. These limitations often require of assistive robotic arms that could be transported from one place to another. The complexity of the assistive robot arms and their work environments creates a need for faster communications (Klebbe, Scherzinger, Eicher, & others, 2022) as they allow to perform computationally intensive models like numerical solutions for inverse kinematics or Internet of Things integration.

## 1.2 Current Solutions and Their Limitations

There is a growing demand for assistive robots and technologies that help the elderly and disabled live independently. In recent years, wheelchair-mounted robotic arms and other physically assistive robotic devices have shown promise in assisting patients with upper limb limitations who are unable to do simple daily chores such as eating and drinking a cup of water autonomously (Tangcharoensathien, Witthayapipopsakul, Viriyathorn, Patcharanarumol, & others, 2018; Toro-Hernández et al., 2019; Valk, Mouton, Otten, & Bongers, 2019). However, in order for these assistive robots to be effective, they must be simple for their users to operate and have an intuitive interface (Craig, Nelson, Li, & Zhang, 2016; Jiang, Wachs, Pendergast, & Duerstock, 2013;

Penkert, Baron, Madaus, Huber, & Berthele, 2021; Y Rabhi, Mrabet, & Fnaiech, 2015; Yassine Rabhi, Mrabet, Fnaiech, & Gorce, 2013).

As for the technology backend of the assistive robots, recent studies have focused on using the EtherCAT field bus, a communication protocol that has a statistically outperforms all the other current methods but at the cost of specialized hardware and having complex stacks (Siciliano, Khatib, & Kröger, 2008). Researchers have developed solutions based on the Linux OS with the PREEMPT_RT patch (Chuang, Yeh, & Yeh, 2021), Xenomai (Hu, Qu, Li, & Zhao, 2022), QNX RTOS (Sygulla et al., 2018), and more recently, in combinations of Windows and Simulink (Cong, Hanh, Phuong, & Duy, 2022; Langlois et al., 2018; Wang, Yang, & Geer, 2021).
As for the software stacks, the most used is commonly Simple Open EtherCAT Master (SOEM) (Hu et al., 2022), TwinCAT (Langlois et al., 2018; Wang et al., 2021), IGH Etherlab, and Acontis (Sygulla et al., 2018). None of the mentioned studies has considered the need for a light and portable computing solution to implement EtherCAT on the go nor apply its technology in assistive robotics.

## 1.3 Proposed Solution

This research proposes a control system for a 6 Degrees of Freedom (DoF) Wheelchair Mounted Assistive Robot for assistance in the Activities of Daily Living (ADLs). The goal is to develop a control platform capable of manipulating an EtherCAT based Assistive Robotic Arm capable of helping its user in three ranges of ADLs: ground, table, and shelf. Hence, the specific aims of this research project are:

- Aim 1: Design of a Real-Time Control Platform (RTCP) capable of controlling EtherCAT robotic actuators and power wheelchairs.
- Aim 2: Integration of the control framework to perform joystick control of both assistive robotic arm and power wheelchair simultaneously.
- Aim 3: Validate the performance of the proposed Real-Time Control System for Wheelchair-Mounted 6DOF Assistive Robotic arms.

The proposed control framework must keep a consistent loop time to maintain the EtherCAT network. Moreover, it must show that the assistive arm can reach and perform ADLs in the three regions. This represents a challenge for the design and validation of the RTCP as it will need to perform forward kinematics, inverse kinematics, individual joint control on the top of keeping a consistent loop time for the network. An unstable RTCP could go from having all the robot joints locked in the error state to an inconsistent displacement of the robot arm compared to the reference positions.

## 1.4 Contribution and Novelties

The novelty of this study lies in three folds. The development of Real-Time Control Platform (RTCP) bringing faster response times and modularity into assistive robotic arms while keeping a light computing unit to deploy the system. The integration of validated novel control framework with joystick modality (Rulik et al., 2022) with a hundred percent successful completion of ADL. And the validation of real-time control over a 6DOF assistive robotic to perform ADLs.

## 1.5 Structure of the Thesis

**Chapter 2: Literature Review** presents the pros and cons of control methods and interfaces for assistive robots and EtherCAT Technology on the matter while it outlines prior research on the subject. It provides the state-of-the-art of assistive robot control to the audience.

**Chapter 3: Real Time Control Platform (RTCP)** explains in detail the major components of the platform and how they are interconnected in the control architecture.

**Chapter 4: EtherCAT Robotic Actuators Control** goes in detail over the components and necessary steps to setup the custom robotic actuators to have them ready for cyclic synchronous control with EtherCAT.

**Chapter 5: 6DOF Assistive Robotic Arm Modeling and Control** presents the forward and inverse kinematics calculations used to determine and control the cartesian position of the robot end effector. After setting up the arm it explains how the platform integrates with the power-wheelchair

**Chapter 7: Experiments, Results and Discussion** this chapter explains the experimental setup to test effectiveness of the EtherCAT network, the control of the robotic actuators and the control of the 6DOF assistive robotic arm. It goes over the results and further discusses about the effectiveness of the control platform.

**Chapter 8: Conclusions and Future Works** summarizes the findings of the study It also discusses potential future projects for this research.

# Chapter 2: Literature Review

## 2.1 State of the Art

Previous studies have investigated adopting control modalities to help people with disabilities handle robotic arms. Among those modalities for controlling assistive devices, such as a powered wheelchair and robotic arms, a physical joystick is extensively used. It can be purchased, is affordable, has a straightforward design, and it can operate the end effector directly (Malkin, Li, Harada, Landay, & Bilmes, 2011). It supports two-dimensional control for the x and y axes, the biggest drawback of currently available commercial joysticks. Researchers used several designs and control strategies to get around this constraint.

Body-machine interface (BMI) was developed in (Ansari, Edla, Dodia, & Kuppili, 2019; Fall et al., 2018, 2015; Francis, Umayal, & Kanimozhi, 2021; Penaloza & Nishio, 2018; Thorp et al., 2015) by utilizing the user's residual motion. The kinematics of the user's neck and shoulder are used to extract the user input, which is then used to create a proportionally controlled signal and drive the power wheelchair. Based on head tilt estimation and an EMG signal, a body machine interface (Baldi, Spagnoletti, Dragusanu, & Prattichizzo, 2017) is displayed. An adaptive head motion system in (Rudigkeit & Gebhard, 2019) enables the user to command both an assistive robotic arm and a computer's visual user interface. (Solea, Margarit, Cernega, & Serbencu, 2019) uses a head pose control algorithm in conjunction with another head gesture detection approach to control a power wheelchair.

A 3D joystick was created (Jiang et al., 2013) to control a robotic arm used as an assistance device. (Y Rabhi et al., 2015) makes a neural network-based proposal for an improved joystick control interface. Gesture control is another method of control that employs hand motions near the wrist for various commands. It utilized Kinect sensors (Ren, Yuan, Meng, & Zhang, 2013), vision sensors (Zhou et al., 2013), ultrasonic sensors (Kalgaonkar & Raj, 2009), etc. for gesture recognition. (Perrin, Cassinelli, & Ishikawa, 2004) uses visual sensor fusion to perform assistive drinking, while (Try, Schöllmann, Wöhle, & Gebhard, 2021) uses laser-based sensors for recognition. This approach uses a variety of algorithms as well as techniques including fuzzy logic (Pulikottil et al., 2018), neural networks (Alonso-Martin & Salichs, 2011), hidden Markov models (Memon, Motan, Akbar, Hameed, & Hasan, 2016), pattern matching (Nishimori, Saitoh, & Konishi, 2007), histograms (Andreasen Struijk, Egsgaard, Lontis, Gaihede, & Bentsen, 2017), and graph matching (Hildebrand et al., 2019).

A voice-controlled powered wheelchair system is also designed to control the assistive technologies and help physically disabled people (Alonso-Martin & Salichs, 2011; Memon et al., 2016; Nishimori et al., 2007; Pulikottil et al., 2018). By using voice commands, the user can operate the wheelchair. Other alternative control methods such as tongue (Andreasen Struijk et al., 2017; Hildebrand et al., 2019), eye tracking (Păsărică, Bozomitu, Cehan, Lupu, & Rotariu, 2015), and the combination were used. Patients who use brain-controlled wheelchairs (BCW) can regain movement function by controlling their wheelchairs with their mind. With the use of a navigation system and a brain-computer interface, this technology will let users operate an electric wheelchair (Puanhvuan, Khemmachotikun, Wechakarn, Wijarn, & Wongsawat, 2017).

Recent publications in robot arm motion control show the advantages of adopting real-time communication technology in the manufacturing field, frequencies of up to 20KHz that allow scaling the complexity of the robot system by adding more sensors (G. Zhang, Ni, Li, & Liu, 2018) or smart actuators (K. Lee et al., 2018). In other cases, its communication speed allows new control methods, as it is kinematic impedance control in the HIWIN commercial robot (Tsai, Tai, Lin, & Chan, 2019).

In mobile robotics, EtherCAT and real-time also have made recent breakthroughs. In earlier studies, researchers proved that EtherCAT could be used for motion control in wheeled mobile robots (Delgado, Kim, You, & Choi, 2016). Following studies built upon it to solve a constant problem of the field: having precise control of the speed while transferring massive amounts within the network (Delgado & Choi, 2019; Delgado, You, & Choi, 2019).

Other studies take EtherCAT into newer fields as it is the software architecture for rescue robots (Y. Lee, Lee, Choi, Park, & Park, 2016), where memory pre-allocation and CPU multi-threading allowed the system to have an average task execution of 400 us. This study questioned and brought considerations into the network topologies used in EtherCAT; if a slave gets damaged, the following actuators in the network will disconnect, an issue that does not happen in CANbus, as an example.

The research of (Paprocki & Erwiński, 2022) addresses that most of the literature presents results on synchronization between EtherCAT masters and slaves. They focus on synchronization mechanisms in EtherCAT (including DC synchronization), Emphasizing servo drives. The authors

proved that high-speed synchronization data exchange (100 us or less) could allow moving the control systems of the drives into the EtherCAT master bringing new control possibilities.
(Kang, 2021) did a study on the implementation of the EtherCAT Master using SOEM stack. Unlike other studies, the real-time performance of the EtherCAT master was measured using a network analyzer at the output interface card so that the analysis includes all possible effects from the EtherCAT master system. The resulting implementation managed to control the system with a 20KHz frequency even in stressed conditions.

This study suggests the implementation of a control platform and methodology without including more sensors or computing power (EtherCAT aside), which would raise the cost and complexity of the system and have other negative effects. It considers both the benefits and drawbacks of the preceding alternatives. The learning curve for new users and current users of a system equivalent to the proposed solution is kept as short as possible, and the implemented control framework is easy to comprehend and integrate with an assistive arm and a wheelchair.

## 2.2 What is Real-Time Computing

It is a term used to describe a system, either hardware, software, or both, that has time/event constraints, and based on it, the system guarantees a response by the established deadline (constrained). The response of this system is currently in the order of microseconds to milliseconds.

## 2.3 How to do Real-Time Computing

Some tools are needed to perform real-time computing and meet its constraints: synchronous programming, custom operating systems, and specialized networks. Here we will investigate the real-time operating systems; later in this document, we will thoroughly inspect the real-time network EtherCAT.

The survey research (Reghenzani, Massari, & Fornaciari, 2019) centered on the PEEMPT_RT Linux Patch; its main advantages include simplicity in its setup as it is a patch over the Linux kernel rather than the setup of secondary kernels (cokernel), higher maintainability and portability. Setting up the PEEMPT_RT patch needs to be done correctly to have realistic latencies and avoid erroneous results on the programs meant to run over the system.

## 2.4 What is CANopen

Before jumping into the EtherCAT field bus, it is advised to review and understand CANbus, specifically CANopen: a standardized embedded network with highly flexible configuration capabilities. We will see a top-down approach to understanding CANopen, starting from the application interface layer.

There are three identifiers to consider in CANopen:

- Node ID to identify a specific node (program); it goes from 1 to 127.
- Object Dictionary Index and Sub Index used to access process data and configuration data (variables of the CAN devices).

- Connection Object (COB) ID is used to identify specific messages on the network. To identify a communication channel between two or multiple nodes.

As for the term object, there are a few definitions to have present:

- Object dictionary that stores the variables and constants of the CAN devices, like a look-up table.
- Process Data Objects (PDOs), the messages/variables that contain process data.
- Services Data Objects (SDOs), the messages/variables that contain configuration/service data.

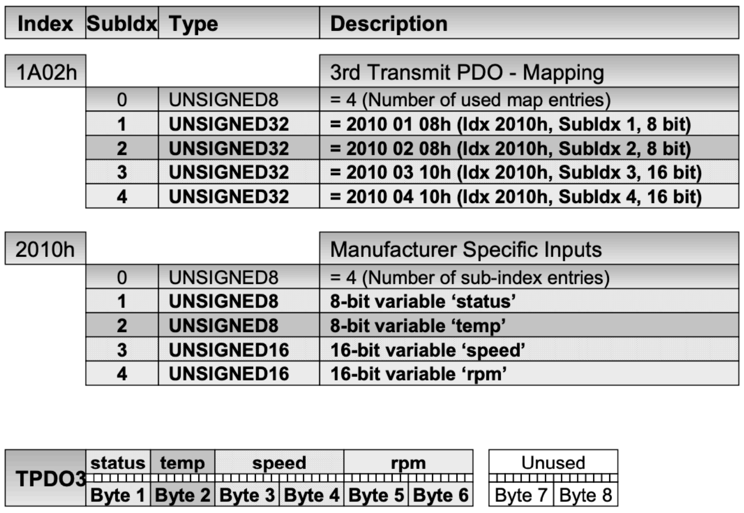
Object dictionaries are organized collections of entries containing several different types of data. The data may be stored in standardized and custom data types (integers, strings, and so), and knowing each variable's type is necessary to extract and send it into a CAN network.

*Table 1 Data Types of Object Dictionaries* (Pfeiffer, Ayre, & Keydel, 2008)

| Standard Data Type | Description | Stored in OD Index |
|---|---|---|
| BOOLEAN | Single bit value 0 or 1 indicating false or true | 0001h |
| INTEGER8 | 8-bit signed integer | 0002h |
| INTEGER16 | 16-bit signed integer | 0003h |
| INTEGER24 | 24-bit signed integer | 0010h |
| INTEGER32 | 32-bit signed integer | 0004h |
| INTEGER40 | 40-bit signed integer | 0012h |
| INTEGER48 | 48-bit signed integer | 0013h |
| INTEGER56 | 56-bit signed integer | 0014h |
| INTEGER64 | 64-bit signed integer | 0015h |
| UNSIGNED8 | 8-bit unsigned integer | 0005h |
| UNSIGNED16 | 16-bit unsigned integer | 0006h |
| UNSIGNED24 | 24-bit unsigned integer | 0016h |
| UNSIGNED32 | 32-bit unsigned integer | 0007h |
| UNSIGNED40 | 40-bit unsigned integer | 0018h |
| UNSIGNED48 | 48-bit unsigned integer | 0019h |
| UNSIGNED56 | 56-bit unsigned integer | 001Ah |
| UNSIGNED64 | 64-bit unsigned integer | 001Bh |
| REAL32 | 32-bit single precision floating point number | 0008h |
| REAL64 | 64-bit double precision floating point number | 0011h |

Now we consider the first step to set up an actuator (slave) on a CAN network by accessing its Object Dictionary using the SDO method. This method is mainly used to set up the configuration variables, the modes of operation, the operation limits, and, more importantly, the map of cyclic variables when the CAN network becomes operational. In this case, an actuator is a node in the network, having its CAN ID and Object Dictionary and acting as a server in which we, as users, can request information or changes on its internal variables using the proper Indexes and Subindexes.

The map of cyclic variables happens in specific Indexes of the object dictionary where the subindex 0 represents the number of variables in the map, and each following subindex stores the "address" of the object/variable to be stored (see Fig. 1).



| Index | SubIdx | Type | Description |
|---|---|---|---|
| 1A02h | | | 3rd Transmit PDO - Mapping |
| | 0 | UNSIGNED8 | = 4 (Number of used map entries) |
| | 1 | **UNSIGNED32** | **= 2010 01 08h (Idx 2010h, SubIdx 1, 8 bit)** |
| | 2 | **UNSIGNED32** | **= 2010 02 08h (Idx 2010h, SubIdx 2, 8 bit)** |
| | 3 | **UNSIGNED32** | **= 2010 03 10h (Idx 2010h, SubIdx 3, 16 bit)** |
| | 4 | **UNSIGNED32** | **= 2010 04 10h (Idx 2010h, SubIdx 4, 16 bit)** |
| 2010h | | | Manufacturer Specific Inputs |
| | 0 | UNSIGNED8 | = 4 (Number of sub-index entries) |
| | 1 | **UNSIGNED8** | **8-bit variable 'status'** |
| | 2 | **UNSIGNED8** | **8-bit variable 'temp'** |
| | 3 | **UNSIGNED16** | **16-bit variable 'speed'** |
| | 4 | **UNSIGNED16** | **16-bit variable 'rpm'** |

| | status | temp | speed | | rpm | | Unused | |
|---|---|---|---|---|---|---|---|---|
| TPDO3 | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 | Byte 8 |

*Figure 1 PDO mapping example (Pfeiffer et al., 2008)*

After the setup done by SDOs, we move to the PDOs, the optimized and intended method to communicate in a CAN-based network. There are two subcategories Transmit PDOs (TPDOs) out of the slave and Receive PDOs (RPDOs) into the slave. Examples include TPDOs (current position, current velocity) and RPDOs (target position, target velocity).

A clear understanding of the network variables and how they are packaged allows us to move into CAN Network Management. It is like a Finite-State Machine. Fig. 2 shows an example CAN state machine of a slave (robotic actuator) where each rectangle represents a state, and each arrow represents a transition. Some transitions could only be triggered automatically by the slave; for example, a motor driver switches to an error state because of an over-current. As other transitions are triggered by a message from the master, for this context, we will call it Control Word, an RPDO that most CAN slaves have.

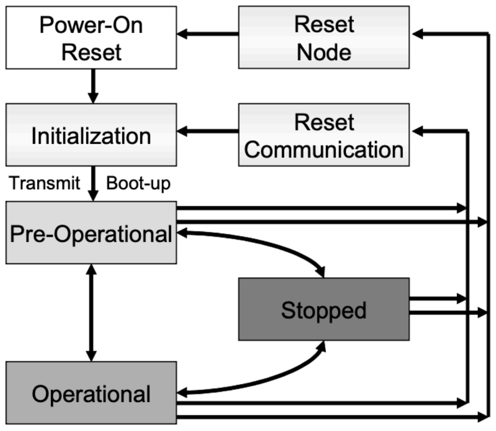


*Figure 2 CANopen slave state machine example* (Pfeiffer et al., 2008)

To know the current position in the CAN state machine, reading the Status Word, an RPDO will return a hex number where each bit represents a different sub-condition.

## 2.5 What is EtherCAT

EtherCAT or Ethernet Control Automation Technology is an industrial standard suitable for hard and soft real-time requirements. Its principle is that the master of the network sends a telegram that passes over each node, where each slave reads the data addressed to it and writes its portion as the frame moves out of it. As the data frame is read/written by each slave, the last link in the network detects that it has an open port and thus proceeds to send the data frame back to the master (see Fig. 3).

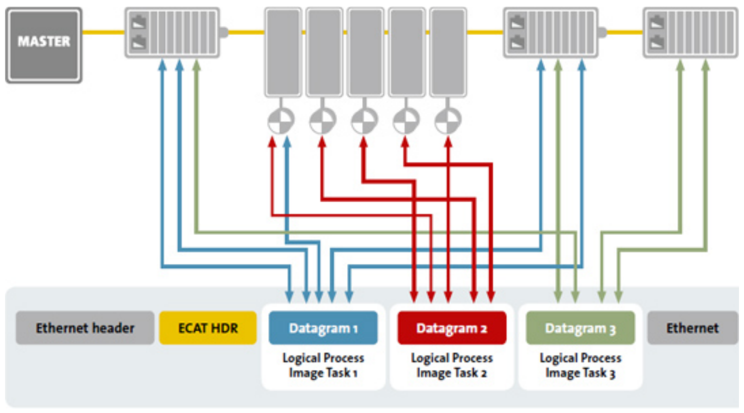

MASTER
Ethernet header
ECAT HDR
Datagram 1
Logical Process Image Task 1
Datagram 2
Logical Process Image Task 2
Datagram 3
Logical Process Image Task 3
Ethernet

*Figure 3 EtherCAT Network diagram (Jansen & Buttner, 2004)*

One of the significant advantages is the addition of Distributed Clocks for High-Precision Synchronization, bringing a high degree of tolerance for jitter in communication. In other words, each slave in the network shares its reference clock to identify the propagation delay that the network might present during the startup. Ideal applications include multiple servo axes executing coordinate movements, which will be part of the aims in the following chapters.

## 2.6 CAN over EtherCAT (CoE)

Another significant advantage of EtherCAT is the CoE communication profile with the exact mechanisms as the CANopen®-Standard with its: Object Dictionaries, PDO Mapping, SDOs, and even Network Management. CoE is essential because all previously mentioned theories and structures are reusable and have the same logic for this communication protocol.

## 2.7 EtherCAT State Machine (ESM)

This state machine runs on the EtherCAT master; depending on its conditions, multiple functions could be or are not available to control a slave in the network. It is through the master that the changes are made. Fig. 4 shows all the traditional states and transitions. We will look at each one of the states in detail for a clearer understanding.

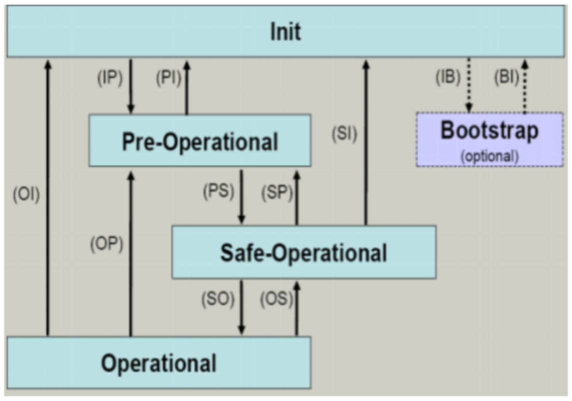


*Figure 4 EtherCAT State Machine (Jansen & Buttner, 2004)*

- Init: After turning the network on, the master and the slaves will be on this state. Process Data Communication is not possible at this point.
- Pre-Operational (Pre-Op): reaching this state validates that the EtherCAT network start was correct. At this point, Sync Manager 2 and 3 are initialized, allowing the configuration of the PDO mapping of the slaves that allow it on the network.
- Safe-Operational (Safe-Op): this stage checks if the sync manager channels for process data and the distributed clocks are initialized correctly. This state also enables cyclical updates of the slave inputs while the outputs are in a safe state.
- Operational (Op): this transition happens when the master sends valid output data to the slaves. Once the systems enter an operational state, the slaves copy the output of the data frame sent by the master.

*Table 2 EtherCAT stack comparison and uses in research*

| **Stack** | **OS** | **Publications** | **Field** |
|---|---|---|---|
| SOEM | Linux, Windows | (Brinkman, Morris, Chen, Sheikh, & Warren, 2021; Hu et al., 2022; Kang, 2021) | Aerospace |
| TwinCAT | Windows | (Langlois et al., 2018; K. Lee et al., 2018; Paprocki & Erwiński, 2022; Tsai et al., 2019; Wang et al., 2021) | Cobots, industrial robot |
| IGH EtherLab | Linux | (Delgado & Choi, 2019; Delgado et al., 2019) | Mobile Robots |
| Acontis | Linux, Windows | (Sygulla et al., 2018; G. Zhang et al., 2018) | Motion Planning |

# Chapter 3: Real-Time Control Platform (RTCP)

Figure 5 shows the network diagram of the study. This chapter cover the first block as it is the RTCP. Chapter 4 goes over the setup and control of the EtherCAT slaves and Chapter 5 explains the integration of the Permobil Power Wheelchair into the system.

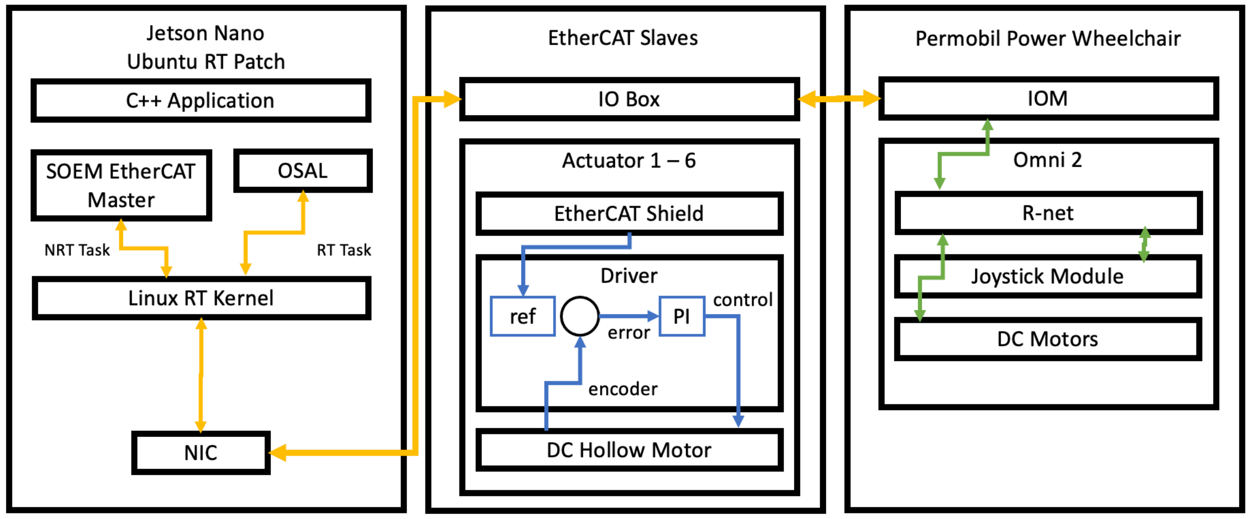


*Figure 5 Software/Network Architecture Diagram*

## 3.1 Hardware

### 3.1.1 Jetson Nano

For this approach, a lightweight, inexpensive, and easy-to-deploy processing unit was considered. Single-board computers fit this criterion exceptionally well, as the Raspberry Pi 4 and Jetson Nano development boards do. Ultimately the Jetson Nano (See Fig. 6.) was selected for its additional GPU. Here are the specifications of the board:

- CPU: Quad-core ARM® A57
- GPU: 128-core NVIDIA Maxwell™ architecture-based GPU

- Memory: 4 GB 64-bit LPDDR4; 25.6 gigabytes/second
- OS: Linux for Tegra® (ARM architecture)
- Network Connectivity: Gigabit Ethernet
- Power Supply: 5V - 4A DC
- Other Connections: 40-pin Header - (3x) I2C, (2x) SPI, UART, I2S, GPIOs

*Figure 6 Jetson Nano Dev. Board*

3.1.2 Cubemars Robotic Actuators

For this study, the Cubemars Custom Hollow DC motor (See Fig. 7) are used. This are servo power units equipped with Field-Oriented-Control algorithm, automatic disturbance rejection and online parameter identification. This motor-driver set is capable of velocity, angle, and torque control. These actuators have a dead weight of 950 gr making them considerably light compared to its closest competitor Qrob (2001 gr). Its precision is of 0.001 rad under a load of up to 40 Nm (Lyy, Zcx, Twy, Cwk, & Jwj, 2022). Here are some of the main specifications (see Appendix P):

- Operating Voltage: 24V
- Maximum continuous current: 20 A
- Communication ports: EtherCAT and CAN

- Resting current: 20 mA
- Dual Encoder Resolution: 14 Bit

*Figure 7 Robotic Actuators* (Lyy et al., 2022)

### 3.1.3 DIEWU EtherCAT Input Output Box

This study requires to have digital inputs and outputs to validate that the system is doing real-time cycles, future studies can use this module to add external input devices as a form of control for the robotic actuators (buttons, joysticks and so). Figure 8 shows the DIEWU EtherCAT IO Box, with 8 Channels, 16 Input, 16 Output, NPN Input Module, 100Mbps with Dual RJ45 Port, AB Phase Encoder.

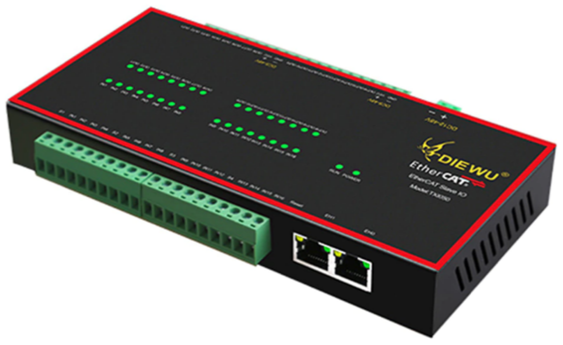


*Figure 8 DIEWU EtherCAT IO Box*

As for the validation of the system response, a Keysight oscilloscope will be used (see Fig. 9) in conjunction with the DIEWU EtherCAT IO Box to generate and read digital signals and measure the loop time of the EtherCAT program running on the master.

*Figure 9 Keysight DSOX1204G Oscilloscope*

### 3.1.4 Permobil M3 Corpus Power Wheelchair

A Permobil M3 Corpus power wheelchair was used to carry out the experiments in this research (Permobil, 2022c) (see Fig. 10a). This powered wheelchair shows a finely tuned product that not only allows its users to smoothly transport from point A to B with an intuitive response to the user input, but it also brings the commodity of adjusting the seat height and tilt to support the needs of its possible users. Thanks to the R-net system used in the wheelchair, multiple modules can be added to expand functionality; for this case, an input/output module was used to extract the user commands from the joysticks into the control computer.

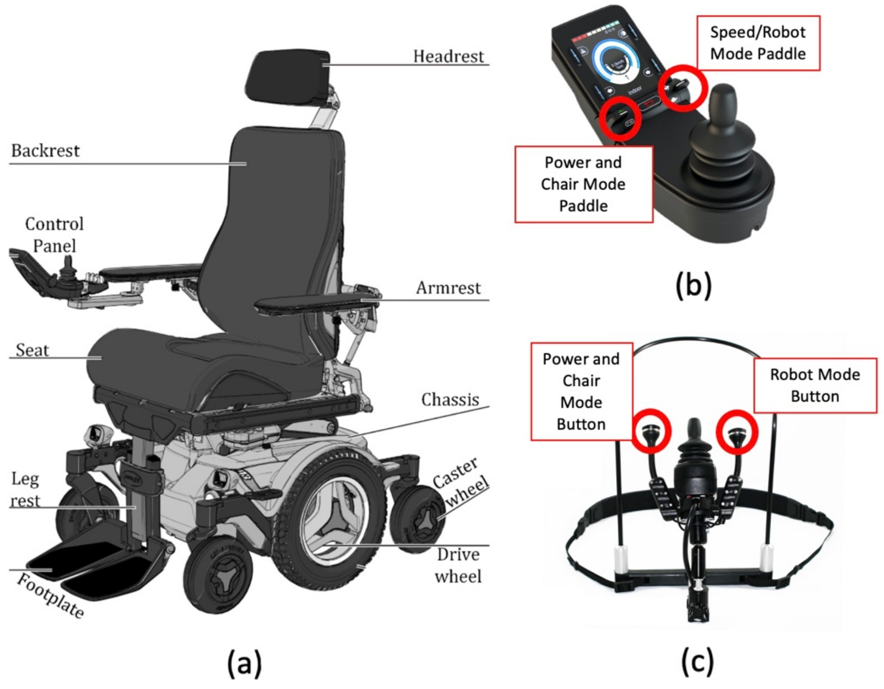


*Figure 10 Permobil M3 corpus, finger joystick and chin joystick* (Rulik et al., 2022)

### 3.1.5 Finger Joystick

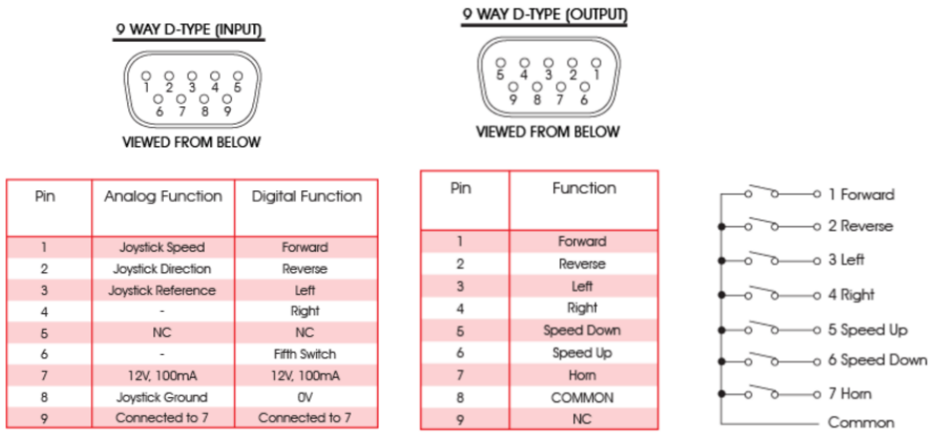
The finger joystick used in this research is shown in Fig. 10b. The joystick used in this control system, developed by Curtiss- Wright and sold by Permobil (Permobil, 2022b), has advantages by offering a two degree of freedom joystick and two control paddles, one on the left to power the system and switch the chair operation modes while the right paddle changes the wheelchair movement speed. It also has four programable buttons. The system has a program that changes between power wheelchair control and assistive arm manipulation through fixed buttons. When the system enters the assistive arm mode, the signals from the joystick will change the arm's position only based on presets of dimensions of movement (X-Y or Y-Z planes, for example). Using the speed paddle, the user can switch between the six possible modes (X-Y axes, Z-Yaw axes, roll-pitch axes, and gripper mode) to manipulate the robot arm and the gripper attached to the end of the arm.

### 3.1.6 Chin Joystick

The chin joystick used in this research is shown in Fig. 10c. Compact chin joysticks (Permobil, 2022a) are used for chin control, with standard proportional force and thrown in a small package. They are also equipped with remote ON/OFF and mode switches. This chin joystick brings the same base functionality as the finger joystick but is adjusted for patients with physical limitations in their hands and arm. It has two degrees of freedom to address the user's desire to manipulate the power wheelchair and the assistive robotic arm. At the same time, the switches attached to its ports allow power ON/OFF the system and move between wheelchair and active arm control. In this research both finger and chin joystick will be considered as a joystick module.

### 3.1.7 Permobil Input Output Module (IOM)

A personalized setup for the powered Wheelchair (Permobil M3 Corpus) was made in which an I/O Module was used between the joystick and the onboard computer. This module has an Output Connector D-Sub 9 Pin (like VGA port) with the following arrangement (see Fig. 11).



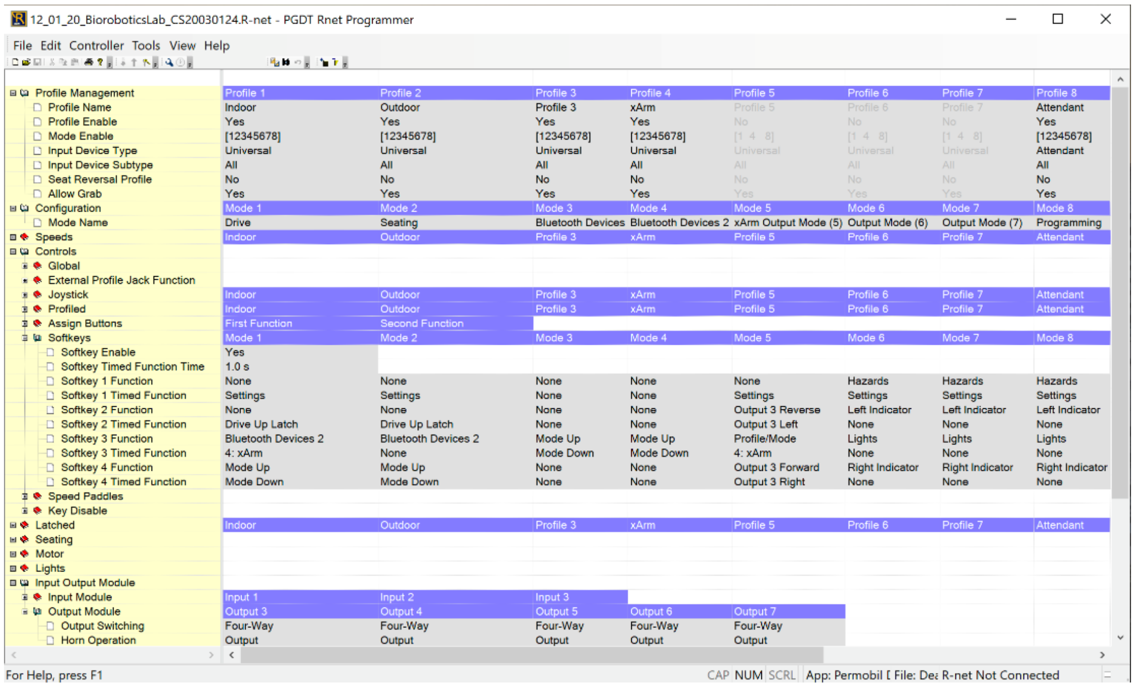
| Pin | Analog Function | Digital Function |
|---|---|---|
| 1 | Joystick Speed | Forward |
| 2 | Joystick Direction | Reverse |
| 3 | Joystick Reference | Left |
| 4 | - | Right |
| 5 | NC | NC |
| 6 | - | Fifth Switch |
| 7 | 12V, 100mA | 12V, 100mA |
| 8 | Joystick Ground | 0V |
| 9 | Connected to 7 | Connected to 7 |

| Pin | Function |
|---|---|
| 1 | Forward |
| 2 | Reverse |
| 3 | Left |
| 4 | Right |
| 5 | Speed Down |
| 6 | Speed Up |
| 7 | Horn |
| 8 | COMMON |
| 9 | NC |

*Figure 11 Input-Output pin layout of the IOM of the power wheelchair* (Curtiss-Wright, 2022)

Before moving into the control computer, we needed to program the onboard computer of the power wheelchair (Permobil M3 Corpus) using the programmer module and the software R-NET in Windows.

This section shows how the onboard computer of the Wheelchair was programmed (see Fig. 12) by creating a robotic arm profile and a mode where the buttons, switches, and joystick will trigger diagonal signals in the output pins of the module. In this case, we adjusted the IOM to be on the Output 3 Configuration of the software.



12_01_20_BioroboticsLab_CS20030124.R-net - PGDT Rnet Programmer

File Edit Controller Tools View Help

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Profile Management | Profile 1 | Profile 2 | Profile 3 | Profile 4 | Profile 5 | Profile 6 | Profile 7 | Profile 8 |
| Profile Name | Indoor | Outdoor | Profile 3 | xArm | Profile 5 | Profile 6 | Profile 7 | Attendant |
| Profile Enable | Yes | Yes | Yes | Yes | No | No | No | Yes |
| Mode Enable | [12345678] | [12345678] | [12345678] | [12345678] | [1 4 8] | [1 4 8] | [1 4 8] | [12345678] |
| Input Device Type | Universal | Universal | Universal | Universal | Universal | Universal | Universal | Attendant |
| Input Device Subtype | All | All | All | All | All | All | All | All |
| Seat Reversal Profile | No | No | No | No | No | No | No | No |
| Allow Grab | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Configuration | Mode 1 | Mode 2 | Mode 3 | Mode 4 | Mode 5 | Mode 6 | Mode 7 | Mode 8 |
| Mode Name | Drive | Seating | Bluetooth Devices | Bluetooth Devices 2 | xArm Output Mode (5) | Output Mode (6) | Output Mode (7) | Programming |
| Speeds | Indoor | Outdoor | Profile 3 | xArm | Profile 5 | Profile 6 | Profile 7 | Attendant |
| Controls | | | | | | | | |
| Global | | | | | | | | |
| External Profile Jack Function | | | | | | | | |
| Joystick | Indoor | Outdoor | Profile 3 | xArm | Profile 5 | Profile 6 | Profile 7 | Attendant |
| Profiled | Indoor | Outdoor | Profile 3 | xArm | Profile 5 | Profile 6 | Profile 7 | Attendant |
| Assign Buttons | First Function | Second Function | | | | | | |
| Softkeys | Mode 1 | Mode 2 | Mode 3 | Mode 4 | Mode 5 | Mode 6 | Mode 7 | Mode 8 |
| Softkey Enable | Yes | | | | | | | |
| Softkey Timed Function Time | 1.0 s | | | | | | | |
| Softkey 1 Function | None | None | None | None | None | Hazards | Hazards | Hazards |
| Softkey 1 Timed Function | Settings | Settings | None | None | Settings | Settings | Settings | Settings |
| Softkey 2 Function | None | None | None | None | Output 3 Reverse | Left Indicator | Left Indicator | Left Indicator |
| Softkey 2 Timed Function | Drive Up Latch | Drive Up Latch | None | None | Output 3 Left | None | None | None |
| Softkey 3 Function | Bluetooth Devices 2 | Bluetooth Devices 2 | Mode Up | Mode Up | Profile/Mode | Lights | Lights | Lights |
| Softkey 3 Timed Function | 4: xArm | None | Mode Down | Mode Down | 4: xArm | None | None | None |
| Softkey 4 Function | Mode Up | Mode Up | None | None | Output 3 Forward | Right Indicator | Right Indicator | Right Indicator |
| Softkey 4 Timed Function | Mode Down | Mode Down | None | None | Output 3 Right | None | None | None |
| Speed Paddles | | | | | | | | |
| Key Disable | | | | | | | | |
| Latched | Indoor | Outdoor | Profile 3 | xArm | Profile 5 | Profile 6 | Profile 7 | Attendant |
| Seating | | | | | | | | |
| Motor | | | | | | | | |
| Lights | | | | | | | | |
| Input Output Module | | | | | | | | |
| Input Module | Input 1 | Input 2 | Input 3 | | | | | |
| Output Module | Output 3 | Output 4 | Output 5 | Output 6 | Output 7 | | | |
| Output Switching | Four-Way | Four-Way | Four-Way | Four-Way | Four-Way | | | |
| Horn Operation | Output | Output | Output | Output | Output | | | |

For Help, press F1 CAP NUM SCRL App: Permobil [ File: Dea R-net Not Connected

*Figure 12 Permobil's R-net programming software in Windows*

As a result, we have a custom connection to the I/O Module of the power wheelchair that has organized individual pins to read the multiple user-generated signals into the control computer, so the next step will be to process that information and control the robotic arm.

## 3.2 Software

### 3.2.1 Ubuntu Real-Time Patch

The platform runs on the Linux Operative System, more Specifically Ubuntu desktop 18.04 (Jetpack) with PREEMPT-RT Patch. To do real-time patch on Ubuntu, the following steps are required:

- Run the command *uname -r* to know which kernel currently has the PC.
- Go to https://www.kernel.org/pub/linux/kernel/projects/rt/ and find the closes kernel with an RT patch, download the one with extension.tar.gz
- Based on the RT patch go to https://www.kernel.org/pub/linux/kernel/ and find the same kernel as the one in the RT patch and download it.
- Run the command *mkdir ~/kernel/* to create a folder on the home of the user. From there copy all the downloaded files into it.
- Run the command *tar -xzvf linux-5.15.79.tar.gz* and then *cd linux-5.15.79* to expand and move into the kernel folder.
- Run the command *gzip -cd ../patch-4.9.115-rt93.patch.gz | patch -p1 –verbose* to paste the patch into the kernel folder.
- Run *sudo apt-get install libncurses-dev libssl-dev flex bison openssl dkms libelf-dev libpci-dev libiberty-dev autoconf* to install all the dependencies required for compiling the kernel.
- Run *make menuconfig* this will show a GUI to create a .config file

- Inside of menuconfig go into General Settings -> Preemption Model -> Fully Preemptible Kernel (RT) and hit enter.
- Use the arrows to select the Save button and the bottom bar and press the enter key to select it.
- Select the Exit option from the bottom part until the GUI is closed.
- Run the command *make -j20*
- Then run sudo make modules_install -j20
- Run sudo make install -j20
- Finally run *cd /boot* and then *sudo update- grub* this will update the available kernels for the OS to pick when it is booting up.

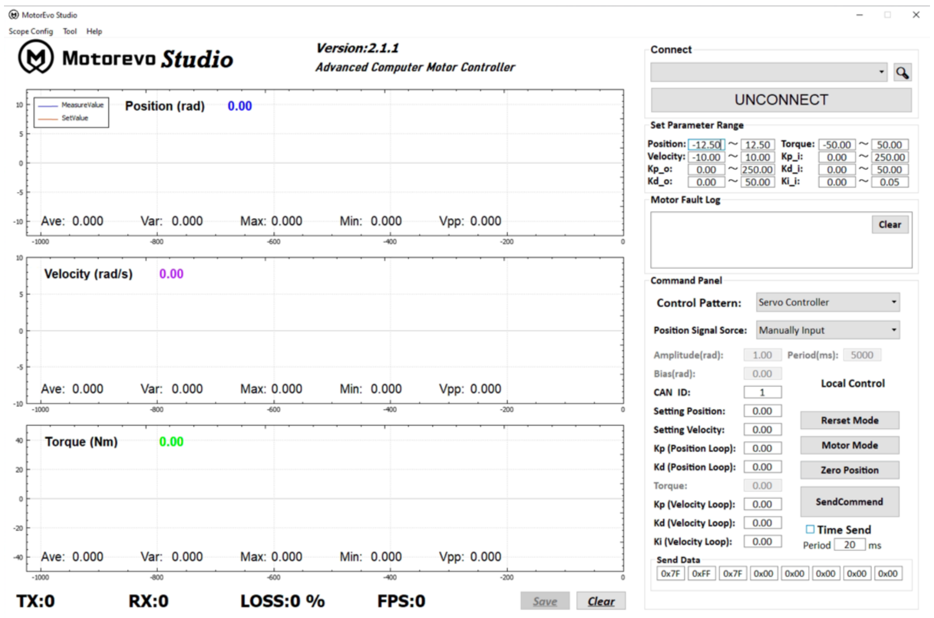
# Chapter 4: EtherCAT Robotic Actuators Control

## 4.1 Preparation of the robotic actuators

### 4.1.1 Cubemars specific software MotorevoStudio

The software provided by the manufacturer is Motorevo-Studio, a windows exclusive program that uses both the USB serial communication and the CAN communication to talk with the actuator drivers and calibrate its encoders and electrical variables. The software can be downloaded from the manufacturers GitHub https://github.com/SupremeLyy/Motorevo-Studio/, for this study the version 2.1.1 beta was used as the manufacturer advised.

In Figure 13 we can see the main window with its components. The main window has graphs to display the motors current position, velocity, and torque; this window also has a CAN control interface to test the response of the motors before fixing them into any system; this tool will not be used or covered since the study is centered in EtherCAT control. Over the next sub sections, we will explain in detail the tools necessary to setup the actuators.



Version:2.1.1
Advanced Computer Motor Controller

Position (rad) 0.00
Ave: 0.000 Var: 0.000 Max: 0.000 Min: 0.000 Vpp: 0.000

Velocity (rad/s) 0.00
Ave: 0.000 Var: 0.000 Max: 0.000 Min: 0.000 Vpp: 0.000

Torque (Nm) 0.00
Ave: 0.000 Var: 0.000 Max: 0.000 Min: 0.000 Vpp: 0.000

TX:0 RX:0 LOSS:0 % FPS:0

Set Parameter Range

| | | | | | |
|---|---|---|---|---|---|
| Position: | -12.50 ~ 12.50 | Torque: | -50.00 ~ 50.00 |
| Velocity: | -10.00 ~ 10.00 | Kp_i: | 0.00 ~ 250.00 |
| Kp_o: | 0.00 ~ 250.00 | Kd_i: | 0.00 ~ 50.00 |
| Kd_o: | 0.00 ~ 50.00 | Ki_i: | 0.00 ~ 0.05 |

Motor Fault Log

Command Panel
Control Pattern: Servo Controller
Position Signal Sorce: Manually Input
Amplitude(rad): 1.00 Period(ms): 5000
Bias(rad): 0.00
Local Control
CAN ID: 1
Setting Position: 0.00
Setting Velocity: 0.00
Kp (Position Loop): 0.00
Kd (Position Loop): 0.00
Torque: 0.00
Kp (Velocity Loop): 0.00
Kd (Velocity Loop): 0.00
Ki (Velocity Loop): 0.00
☐ Time Send Period 20 ms
Send Data 0x7F 0xFF 0x7F 0x00 0x00 0x00 0x00 0x00

*Figure 13 Motorevo-Studio overview*

## 4.1.2 Actuator connection

To power on the motors, we used a 24V 60A power supply. If the status buzzer and LED don't show any sign of errors, we connect the motor to a Windows 10 PC using the USB serial port of the motor's driver. Figure 14. shows the back of the actuator with its connectors labeled.

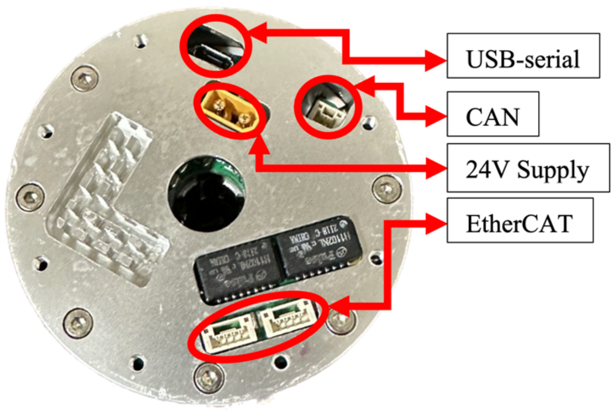
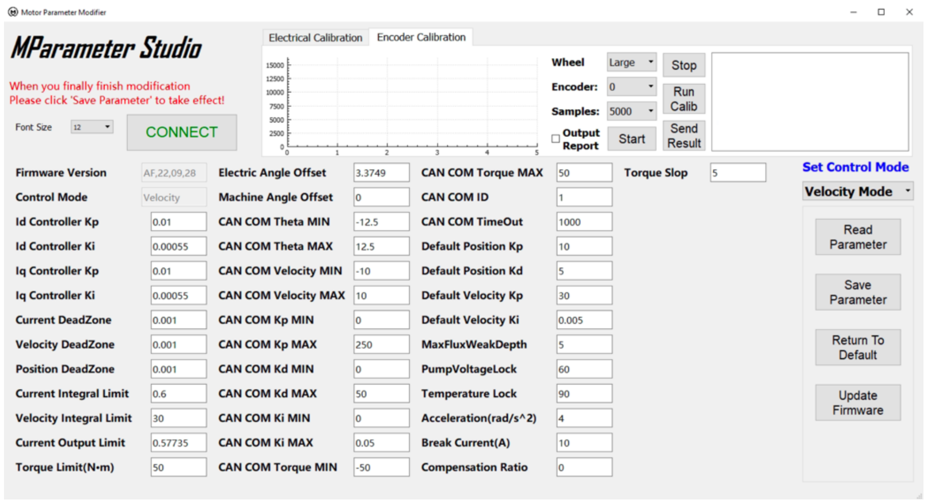


*Figure 14 Robotic Actuator connectors*

After a successful connection of the motor, we launch the Motor Parameter Modify, from the Tool bar of Motorevo-Studio; this tool allowed to calibrate the motors encoders and electrical parameters. The Figure 15 shows the window with the motor's parameters which include the constants for the two control loops, limits of current, torque, velocity and so.



When you finally finish modification
Please click 'Save Parameter' to take effect!

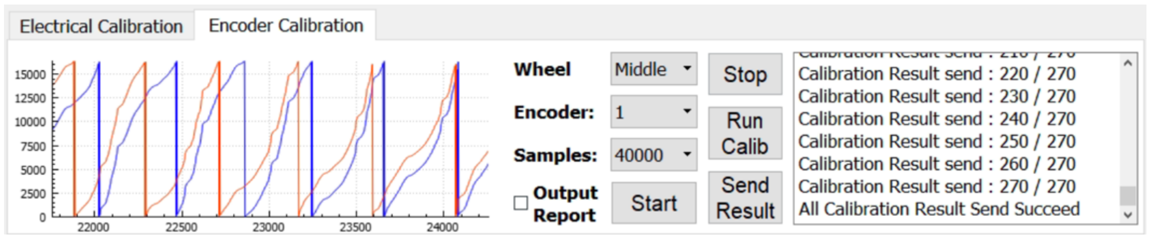
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Font Size | 12 | | | Wheel | Large | | |
| | | | | Encoder: | 0 | | |
| | | | | Samples: | 5000 | | |
| | | | | ☐ Output Report | | | |
| Firmware Version | AF,22,09,28 | Electric Angle Offset | 3.3749 | CAN COM Torque MAX | 50 | Torque Slop | 5 |
| Control Mode | Velocity | Machine Angle Offset | 0 | CAN COM ID | 1 | | Velocity Mode |
| Id Controller Kp | 0.01 | CAN COM Theta MIN | -12.5 | CAN COM TimeOut | 1000 | | |
| Id Controller Ki | 0.00055 | CAN COM Theta MAX | 12.5 | Default Position Kp | 10 | | |
| Iq Controller Kp | 0.01 | CAN COM Velocity MIN | -10 | Default Position Kd | 5 | | |
| Iq Controller Ki | 0.00055 | CAN COM Velocity MAX | 10 | Default Velocity Kp | 30 | | |
| Current DeadZone | 0.001 | CAN COM Kp MIN | 0 | Default Velocity Ki | 0.005 | | |
| Velocity DeadZone | 0.001 | CAN COM Kp MAX | 250 | MaxFluxWeakDepth | 5 | | |
| Position DeadZone | 0.001 | CAN COM Kd MIN | 0 | PumpVoltageLock | 60 | | |
| Current Integral Limit | 0.6 | CAN COM Kd MAX | 50 | Temperature Lock | 90 | | |
| Velocity Integral Limit | 30 | CAN COM Ki MIN | 0 | Acceleration(rad/s^2) | 4 | | |
| Current Output Limit | 0.57735 | CAN COM Ki MAX | 0.05 | Break Current(A) | 10 | | |
| Torque Limit(N•m) | 50 | CAN COM Torque MIN | -50 | Compensation Ratio | 0 | | |

*Figure 15 Actuator parameters in Motorevo-Studio*

To validate the correct connection and work of the motor it is needed to click Connect and Read Parameters. If the window fills all or most of the fields, it means that the motor is ok, and the calibrations steps can be done. The next sub sections will go over the calibration of the motor encoder for precise position reading and electrical calibration for a precise control of torque, velocity, and position.

## 4.1.3 Calibration of Encoders

The calibration of the encoders is a necessary process before doing anything else with the robotic actuators since it minimizes the error between a measured position and a real position. Wrong calibrated encoders could represent a robot arm breaking its own joints or an electric vehicle stopping a few inches or meter after a stop sign. Figure 16. shows visual feedback of the encoder reading while a calibration signal is sent. As advised by the manufacturer the program should take the calibration samples for approximately three minutes before uploading the new configuration.



```
Calibration Result send : 220 / 270
Calibration Result send : 230 / 270
Calibration Result send : 240 / 270
Calibration Result send : 250 / 270
Calibration Result send : 260 / 270
Calibration Result send : 270 / 270
All Calibration Result Send Succeed
```

*Figure 16 Calibration feedback encoders*

### 4.1.4 Electrical Calibration of the Motor

The electrical calibration goes over a phase check where the driver determines the which motor winding was connected to which pin of the board; based on that result the program reconfigures the driver and follows to the next stage. At the calibration stage the program sends a reference position and plots the real position of the motor, as seen in Fig. 17. After the calibration stage is finished the program writes the new parameters into the motor driver. The manufacturer advised to read the parameter again and check that the Electrical Offset Angle changes as this compensates for the difference between the reference and the measured signals of the actuator.

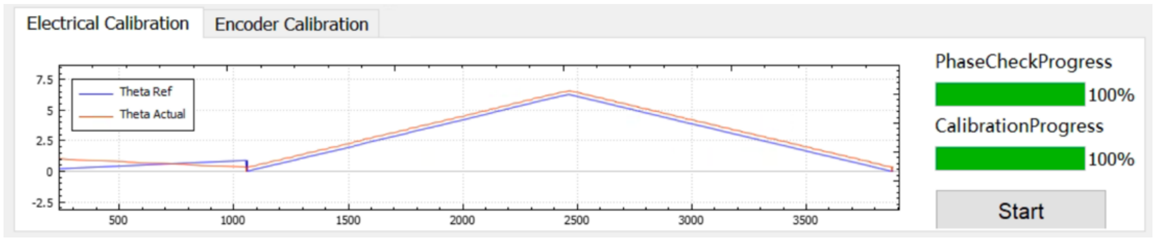


*Figure 17 Calibration feedback motor*

After a successful calibration, the next step was to prepare the experimental setup to test the response of the actuators.

## 4.2 Experimental Setup

The experimental setup includes a 24V 60A power supply, the Jetson Nano as EtherCAT Master, the EtherCAT IO Box as the first slave and the rest of the calibrated motor as the following slaves. Additionally, the experimental setup includes an oscilloscope connected to the digital output of the IO Box to measure the cycle time of the whole system. The connection of the oscilloscope took an extra step since the output of the IO Box uses a sink logic and thus a 5-volt supply and a protection resistor were added to have a measurable read. All the network connections were made using regular RS45 cables and connectors. Finally, to program and control the EtherCAT master a mouse, keyboard and display were connected to it. Figure 18 shows the experimental setup.

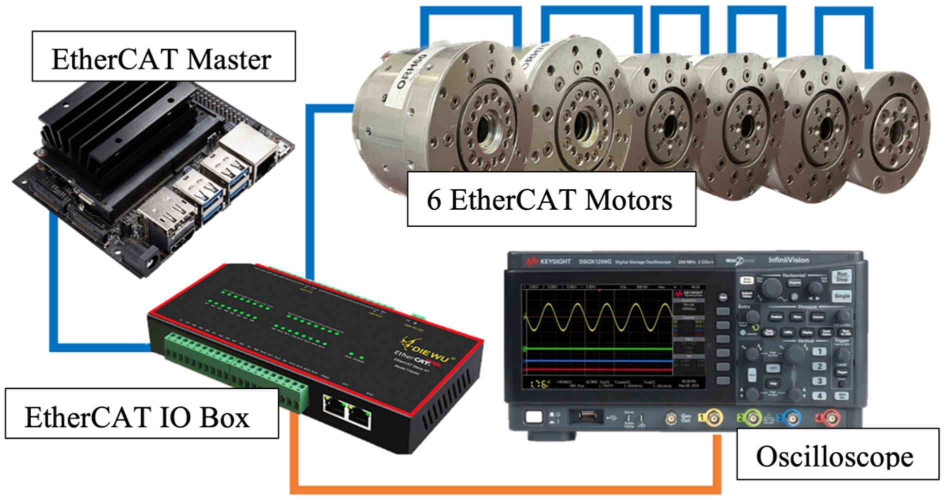


*Figure 18 Experimental Setup Single Actuator*

## 4.3 EtherCAT Network Setup

In this sub-section we test the network connections and capabilities of the experimental setup and from that point we determine the PDO map to be loaded into the network slaves. Finally, we check that the PDO map was loaded properly into the system.

### 4.3.1 EtherCAT Network Test

To test the EtherCAT network in the experimental setup, a SOEM script is used. The code slave info is a built-in program in SOEM library that reads the network and returns a small report of each one of the slaves connected. The network report includes:

- Input Size
- Output Size
- State
- Delay in ns (for the EtherCAT buffer to reach the specific slave)

When the program slaveinfo is called, an additional switch (-map) can be used to get the current PDO maps loaded into the drivers. The next subsection will go into more detail.

### 4.3.2 EtherCAT PDO Mapping

The following table shows the current PDO mapping used for all the robotic actuators (Cubemars Motors). The PDO mapping table includes all the necessary information required to program and load the PDOs using C++ and SOEM. Table 3 shows the PDO variables loaded into the actuators of the test bench.

*Table 3 EtherCAT Slave (Drive) PDO Mapping Data.*

| Index | Size | Name | Type |
|---|---|---|---|
| Transmit PDO Map | | | |
| 0x1600:00 | 4.0 | TPDO map 1 index | UINT8 |
| 0x1600:01-n | 4.0 | TPDO map 1 slot 1 to n | UINT32 |
| 0x6040:00 | 2.0 | Control word | UINT 16 |
| 0x607A:00 | 4.0 | Target Position | INT 32 |
| 0x60FF:00 | 4.0 | Target Velocity | INT 32 |
| 0x6077:00 | 4.0 | Target Torque Actual | INT 32 |
| 0x6040:00 | 2.0 | Modes of Operation | INT 8 |
| Receive PDO Map | | | |
| 0x1A00:00 | 1.0 | RPDO map 1 index | UINT8 |
| 0x1A00:01-n | 4.0 | RPDO map 1 slot 1 to n | UINT32 |
| 0x6040:00 | 2.0 | Status Word | INT 16 |
| 0x6064:00 | 4.0 | Position Actual Value | INT 32 |
| 0x606C:00 | 4.0 | Velocity Actual Value | INT 32 |
| 0x6077:00 | 4.0 | Torque Actual Value | INT 32 |
| 0x6040:00 | 2.0 | Modes of Operation Display | INT 8 |

## 4.4 Real-time Tests with Multiple Modes of Operation

Cyclic synchronous communication refers to the continuous transmission of messages at fixed times, with the master and drive time-synchronized in accordance with a set protocol.

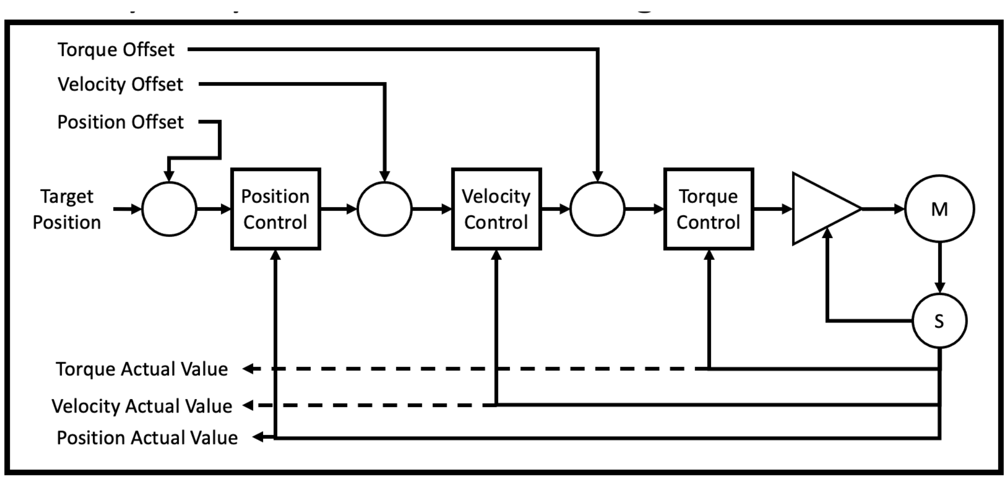


*Figure 19 Cyclic Synchronous Control Diagram*

The Figure 19 shows the diagram for cyclic synchronous control based on the CAN standard (Pfeiffer et al., 2008). The following subsections will go in more detail over each mode used and how to use them in an EtherCAT network.

The steps to set up this mode of operation goes as follows:

- Set the EtherCAT network into operational mode
- Set the mode of operation PDO to 0x000A (CST) or 0x0008 (CSP)
- Set the command word PDO to 0x0080 to remove any error from the target motor
- Set the command word PDO to 0x0000 to disable the target motor
- Set the command word PDO to 0x0006 to prepare the target motor to power
- Set the command word PDO to 0x0007 to switch on the target motor
- Set the command word PDO to 0x000F to set operational the target motor

From this point modifying the target PDO with any positive or negative value will move the motors

To stop the network set the command word PDO to 0x0007 to return the motor to an idle state

Finally stop the EtherCAT Network and close the code

### 4.4.1 Cyclic Synchronous Torque (CST) Mode

In this mode the drive closes the torque loop without considering the actual velocity or position after receiving a target torque value from the master. Both the maximum torque and the maximum motor speed may be set using CST control. For situations where the load varies and real-time servo loop gain adjustment is required, the cyclic synchronous torque mode is ideally suited.

### 4.4.2 Cyclic Synchronous Velocity (CSV) Mode

the master sends a target velocity to the drive-in accordance with the PDO update cycle, based on the actual motor velocity and torque. The output of the velocity control loop is an input for the torque control loop, and velocity and torque offsets may also be sent to the drive for feed-forward control. With CSV mode, the maximum motor velocity and “quick stop” functions can be defined. Cyclic synchronous velocity mode is often used for speed-control applications, such as spindles.

### 4.4.3 Cyclic Synchronous Position (CSP) Mode

the master sends a target position to the drive-in accordance with the PDO update cycle, based on the actual motor position and, in some cases, velocity and torque as well. The drive closes the control loops, which can be simple PID control or cascaded control with position, velocity, and torque loops. Limit functions can also be defined in CSP mode to prevent the motor from moving to beyond the intended range. Cyclic synchronous position mode is often used in applications with distributed control.

# Chapter 5: 6 DOFs Assistive Robotic Arm Modeling and Control

## 5.1 Denavit-Hartenberg and Link Assignment

The 6 DOF assistive robotic arm shown here was the result of a collaboration with UW-Milwaukee Biorobotics Laboratory. The assistive robotic arm is composed by six active revolute joints (Cubemars). Figure 20 shows the Link-frame alignments while Table 4 shows the Modified Denavit-Hartenberg (DH) Parameters of the 6 DOF assistive arm. The parameters are used in the homogenous transformation matrix (HTM) as it can be seen here:

$$
{}^{i-1}T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \cos\alpha_{i-1}\sin\theta_i & \cos\alpha_{i-1}\cos\theta_i & -\sin\alpha_{i-1} & -\mathrm{d_i}\sin\alpha_{i-1} \\ \sin\alpha_{i-1}\sin\theta_i & \sin\alpha_{i-1}\cos\theta_i & \cos\alpha_{i-1} & d_i\cos\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}
$$

The term $a_{i-1}$ is the distance from the X-axis of the frame i to the X-axis of the frame i-1; $\alpha_{i-1}$ is the angle between the Z-axis of the frame i and the Z-axis of the frame i-1; $d_i$ is the distance from the Z-axis of the frame i to the Z-axis of the frame i-1; and $\theta_i$ is the angle between the X-axis of the frame i and the X-axis of the frame i-1; this definitions follow the modified DH convention as presented in (Siciliano et al., 2008).

*Table 4 Modified Denavit-Hartenberg (DH) Parameters for 6 DOF Assistive Robotic Arm*

| i | $\alpha_{i-1}$ | $a_{i-1}$ | $d_i$ | $\theta_i$ |
|---|---|---|---|---|
| Joint 1 | 0 | 0 | $L_1$ | $\theta_1$ |

| | | | | |
|---|---|---|---|---|
| Joint 2 | $-\pi/2$ | 0 | 0 | $\theta_2 - \pi/2$ |
| Joint 3 | $\pi$ | $L_2$ | 0 | $\theta_3 - \pi/2$ |
| Joint 4 | $-\pi/2$ | 0 | $L_3 + L_4$ | $\theta_4$ |
| Joint 5 | $-\pi/2$ | 0 | 0 | $\theta_5$ |
| Joint 6 | $\pi/2$ | 0 | $L_5$ | $\theta_6$ |

Table 5 shows the physical dimension of the arm used in the Modified DH parameters where $L_i$ is the distance between the current joint (i) and the previous one.

*Table 5 Dimensional parameters of the Assistive Robotic Arm*

| Parameter | $L_1$ | $L_2$ | $L_3$ | $L_4$ | $L_5$ |
|---|---|---|---|---|---|
| Dimension (mm) | 170 | 440 | 450 | 100 | 140 |

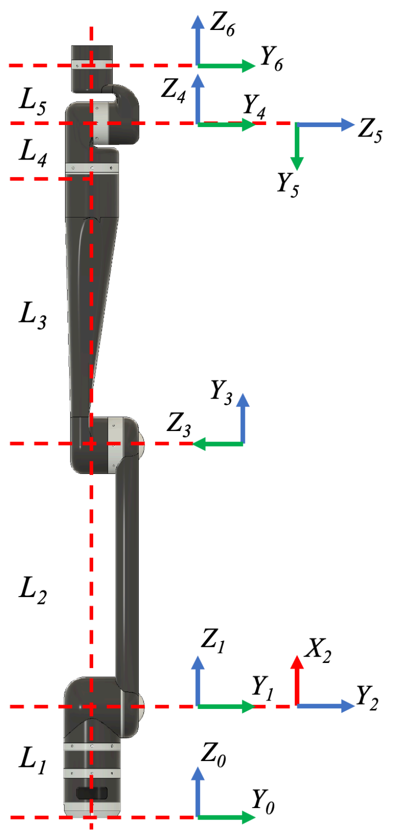


*Figure 20 Link Assignment for 6-DOF Assistive Robotic Arm*

## 5.2 Direct Kinematics

Given the angle of each revolute joint, the direct kinematics of the assistive arm computes the end-effector's posture (position and orientation). Thus, the HTM can and is concatenated to determine the end-effector position and orientation based on all the frames/ joints that came before it, as follows:

$$^{0}T_6 = \ ^{0}T_1 \ ^{1}T_2 \ ^{2}T_3 \ ^{3}T_4 \ ^{4}T_5 \ ^{5}T_6 \tag{2}$$

Furthermore, the end-effector's HTM can be described with the position of the end-effector ${}^{0}P_6$, a column vector, and the orientation matrix ${}^{0}R_6$, constructed using the Euler angles ($\theta_x$, $\theta_y$, $\theta_z$). Thus, ${}^{0}T_6$ represents the direct kinematics of the assistive arm end-effector.

$$
{}^{0}T_6 = \begin{bmatrix} & {}^{0}R_6 & & {}^{0}P_6 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & & & x \\ & R_x(\theta_x)R_y(\theta_y)R_z(\theta_z) & & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}
$$

## 5.3 Inverse Kinematics

Given the posture of the end-effector ${}^{0}T_6$, the inverse kinematics can compute the angle of each joint. However, this process requires solving a set of non-linear equations that involves several trigonometric functions. In literature, many researchers have studied and presented various methods to solve the problem, such as numerical approaches (Li et al., 2021; Starke, Hendrich, & Zhang, 2018; Yiyang, Xi, Hongfei, Zhining, & Liangliang, 2021), algebraic methods (Fu, Yang, & Yang, 2013; Xin, Feng, Bing, & Li, 2007; Zhao et al., 2018), geometrical approaches (Tolani, Goswami, & Badler, 2000; J. Zhang, Zhang, & Zhang, 2021), etc. The inverse kinematics for the Assistive Arm is as follows:

$$
{}^{0}P_5 = {}^{0}T_6 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} - (L_5 + L_6)\, {}^{0}T_6 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{4}
$$

$$
\theta_1 = \arctan\left(\frac{{}^{0}P_{5,y}}{{}^{0}P_{5,x}}\right) \tag{5}
$$

$$ {}^{1}P_{5} = {}^{0}T_{6}^{-1}\ {}^{0}P_{5} \tag{6} $$

$$ \theta_2 = \frac{\pi}{2} - \arctan\left(\frac{{}^{1}P_{5,y}}{{}^{1}P_{5,x}}\right) - \arccos\left(\frac{L_2^2 + {}^{1}P_{5,x}^2 + {}^{1}P_{5,z}^2 - (L_3 + L_4)^2}{2L_2\sqrt{{}^{1}P_{5,x}^2 + {}^{1}P_{5,z}^2}}\right) \tag{7} $$

$$ {}^{2}P_{5} = {}^{1}T_{2}^{-1}\ {}^{1}P_{5} \tag{8} $$

$$ \theta_3 = \arctan\left(\frac{-\,{}^{2}P_{5,y}}{{}^{2}P_{5,x}}\right) \tag{9} $$

$$ {}^{3}P_{5} = {}^{3}T_{4}^{-1}\ {}^{2}P_{5} \tag{10} $$

$$ \theta_4 = \arctan\left(\frac{-\,{}^{3}P_{5,y}}{{}^{3}P_{5,x}}\right) \tag{11} $$

$$ {}^{4}P_{5} = {}^{3}T_{4}^{-1}\ {}^{3}P_{5} \tag{12} $$

$$ \theta_5 = \arctan\left(\frac{{}^{4}P_{5,y}}{{}^{4}P_{5,x}}\right) \tag{13} $$

$$ {}^{5}P_{6} = \left({}^{0}T_{1}\ {}^{1}T_{2}\ {}^{2}T_{3}\ {}^{3}T_{4}\ {}^{4}T_{5}\right)\ {}^{0}T_{6}\begin{bmatrix}1\\0\\0\\0\end{bmatrix} \tag{14} $$

$$\theta_6 = \arctan\left(\frac{{}^{5}P_{6,y}}{{}^{5}P_{6,x}}\right) \tag{15}$$

## 5.4 6DOF Robot Arm Setup

The figure 21 shows the experimental setup of the assistive robotic arm (left) and the power wheelchair with the arm attached to it (right). Recalling chapter 3's figure 12 of the network diagram, we find in this setup the user input devices such as the joystick module. It connects to the power wheelchair Permobil M3 Corpus through the R-net system (Curtiss, 2022) for the transportation control and status display. Over the input/ output module (IOM) (Curtiss-Wright, 2022), the R-net system sends and receives control signals from the control computer. The control computer (RTCP) is where multiple C++ threads run to coordinate the control of the power wheelchair and assistive robotic arm as well, a defining characteristic of this platform is the digital input connectors that physically connect the assistive robot arm with the power wheelchair. In conclusion, there is a robotic arm section, which takes both power and control signals from the computer section and returns the status of the arm. This structure is composed of the brushless DC motors with reduction boxes and motor drivers over a EtherCAT communication; the cyclic synchronous loop is contained in each driver.

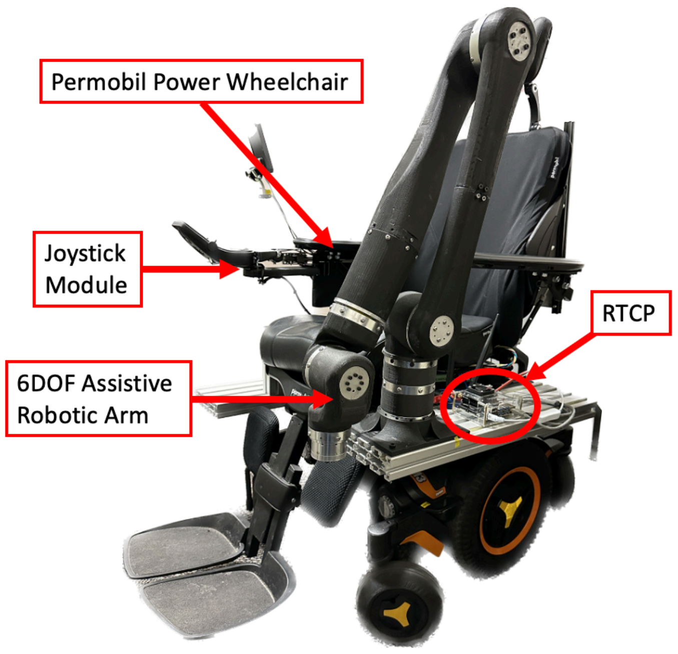


*Figure 21 6-DOF Robot Arm attached to Permobil Power Wheelchair*

The setup is an interconnected system with custom circuitry (see Fig. 22) and code that seamlessly lets the user manipulate a wheelchair and an assistive robot arm. The C++ function for the IO uses the dedicated input pins of the DIEWU IO box to control the robot arm based on the user signals of the joystick module giving individuals with restricted mobility the possibility of performing some ADL. In this diagram VCC is the voltage supply and it comes from the robot supply (24V). The configuration uses pull-up resistors because R-net's IOM has sinking current output pins creating the need to have a regulated voltage on the pin to generate/detect a change.

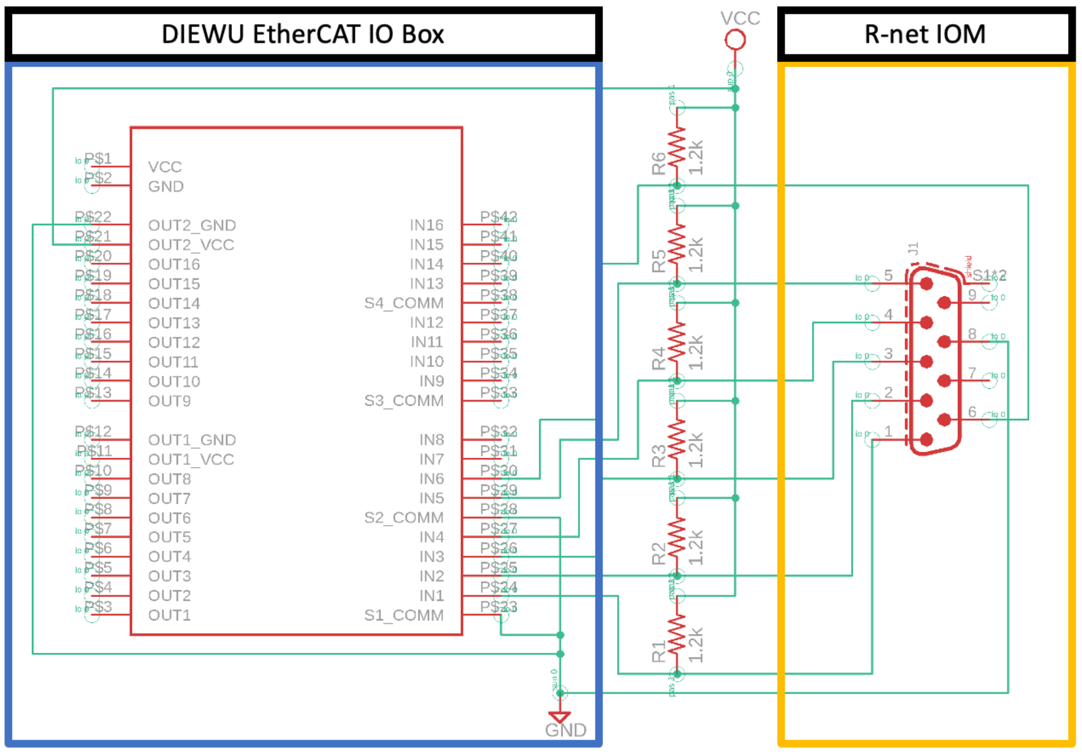


*Figure 22 Wiring diagram of custom IO circuit*

## 5.5 Control of 6DOF Robot Arm

Figure 23 displays the flowchart of the function to manipulate the assistive robotic arm's end-effector position. Once the user starts the robotic system, it will preload the first operation mode, where the assistive arm can be moved on the X–Y axes. Movements along the X–Y axes are forward (Fwd), backward (Bwd), left (Lft), and right (Rght), which can be achieved by selecting the joystick operation mode in the GUI. The user will change the operation modes to access the three- dimensional displacement changing from X–Y to Z direction (up/down), the three rotational movements (roll, pitch, and yaw), and the control over the gripper. To change the acxis of control, the user needs to either move the speed paddle up (SpUp) or down (SpDwn).

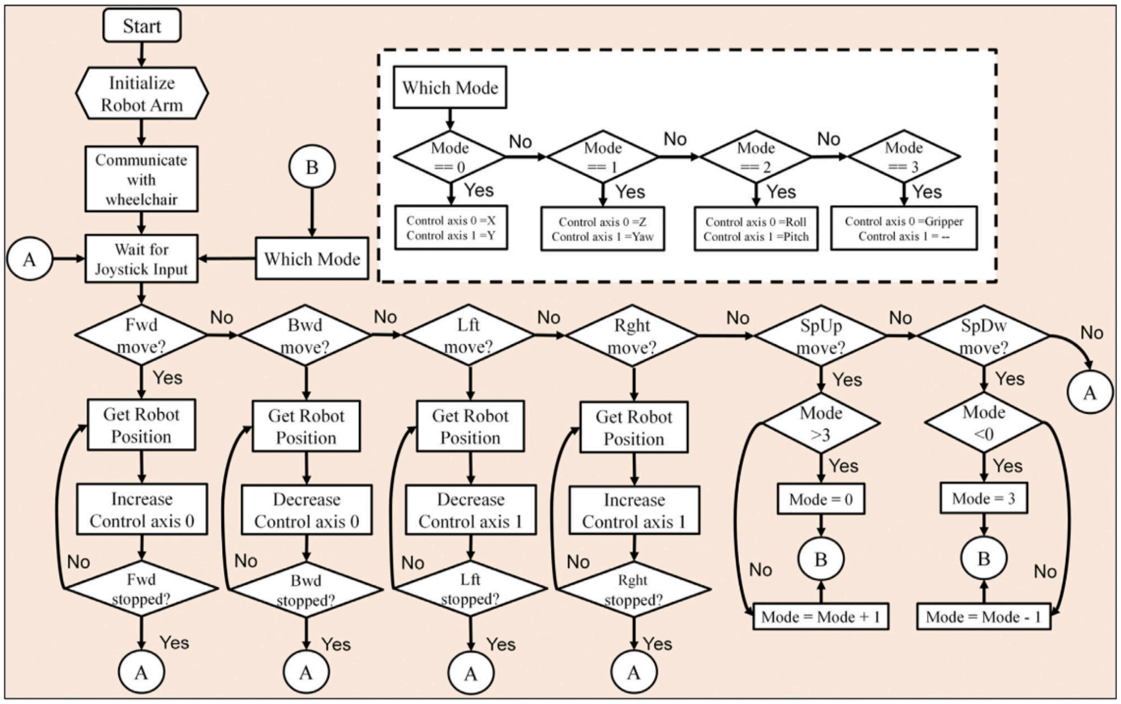


*Figure 23 Flowchart of 6DOFend-effector control* (Rulik et al., 2022)

# Chapter 6: Simple Open EtherCAT Master (SOEM) Control Code

Simple Open EtherCAT Master is the preferred EtherCAT Master Stack (SOEM). All its examples are written for Linux, its primary target OS. However, because each application is unique, SOEM tries not to impose any specific design architecture. Since SOEM runs in user rather than kernel space, so it can be executed on a wide range of operating systems and computer architectures. It can be used in PREEMPT RT, Xenomai, or general user mode. Additionally, it is relatively easy to convert it to various targets. Thus, in this study, it is deployed in a single-board computer running Linux for ARM. The developed program runs on a custom open-source C++ library. The operating system abstraction layer (OSAL) API is used to create real-time threads.

The program was written in the C/C++ language. The most relevant structures and functions used in the program will be displayed and explained as they can be seen in Fig. 24.

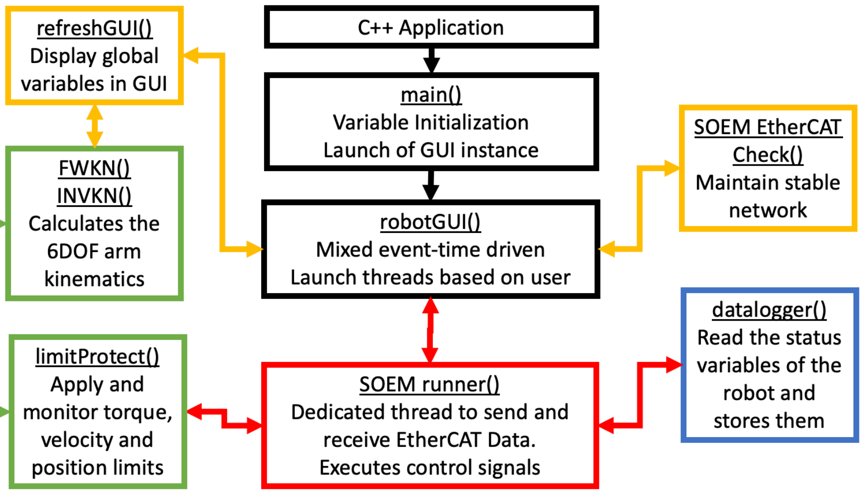

refreshGUI()
Display global variables in GUI
C++ Application
main()
Variable Initialization
Launch of GUI instance
SOEM EtherCAT Check()
Maintain stable network
FWKN()
INVKN()
Calculates the 6DOF arm kinematics
robotGUI()
Mixed event-time driven
Launch threads based on user
datalogger()
Read the status variables of the robot and stores them
limitProtect()
Apply and monitor torque, velocity and position limits
SOEM runner()
Dedicated thread to send and receive EtherCAT Data.
Executes control signals

*Figure 24 Base Program Structure*

## 6.1 Variables and Considerations in the Code

Before going further into the code, there are considerations that need to be mentioned to make the code more understandable:

- The manufacturers of the robotic actuators have disclosed how their drivers both receive and return the network variables as it will be shown (Lyy et al., 2022):
    - Actual Position / Target position: 1*10000[rad]
    - Actual Velocity / Target Velocity: 1*10000[rad/s]
    - Actual Torque / Target Torque: 1*10000[Nm]

  In consequence in multiple parts of the code there will be a conversion as it is more useful for the user to read and send rad, rad/sec and Nm values rather than ten thousand times the value.
- Time control variables like ctime and cycle time are in the microsecond units (us) even thought that these variables are converted to nano seconds (ns) to be used with the internal Linux functions to measure compare and correct the cycle times of the system.
- There are three types of global variables that are used across all functions to share data between them, and they are:
    - Double arrays: these global variables store either the converted position, velocity, or torque of each robot joint; and they are used to store the limits for position, velocity, and torque as well.
    - Unsigned Int arrays: this is the type of variables that the raw PDOs of the EtherCAT network expect for control and return as feedback.
    - Int and Boolean: both variable types are used as counters or control flags to determine if a mode of operation its going. As an example, if the data logger is active there will be a global variable equal to TRUE. Another example is the number of loops that has occurred since a command has been sent, this is stored in a count variable to guarantee a delay without putting the whole system in a halt.

## 6.2 Main Function

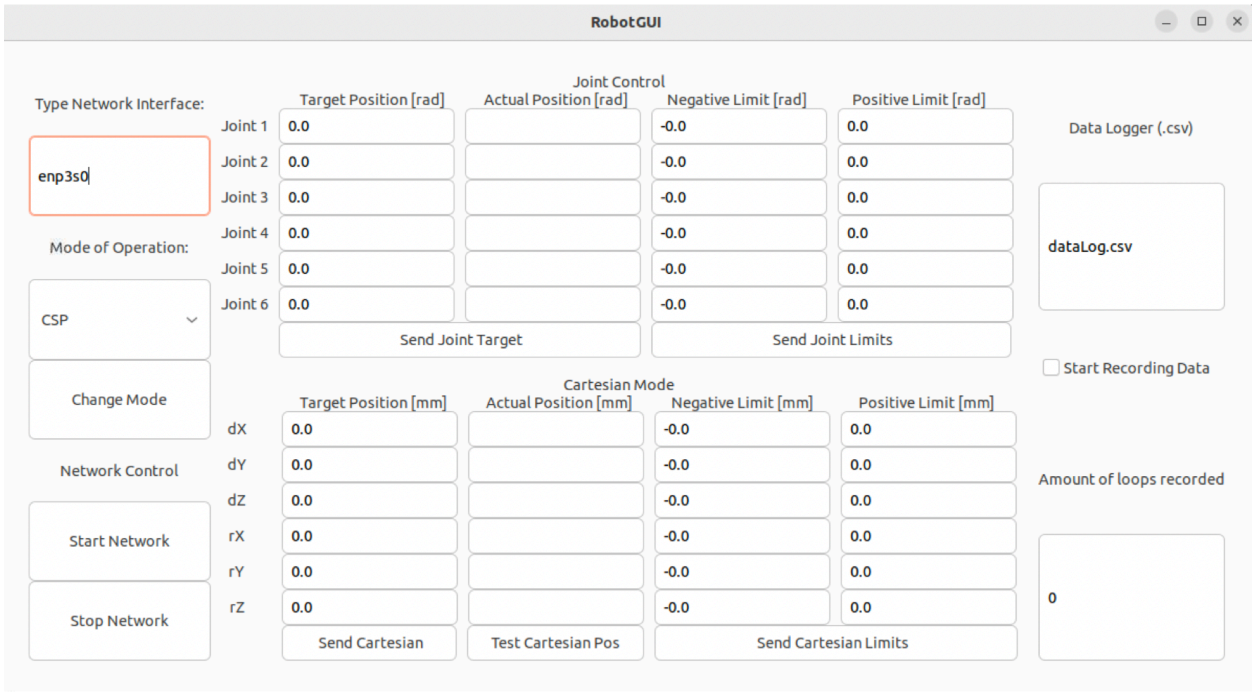
The main function initializes both local and global variables of the system and then it calls a custom instance of the Gtk window class (See Appendix A).

## 6.3 Robot GUI

This instance handles multiple responsibilities:

- Starts a window refreshing thread that will update the network variables on the screen each second (see Appendix B).
- Takes the reference of the Network Interface that will be used for EtherCAT (the computer's Ethernet port) and launches two threads: EtherCAT Master and runner (see Appendix C).
- Handles the control of the revolute joints of the robotic arm, allowing to switch modes of operation and send target values for the robot to follow (see Appendix D and E).
- Allow the user to start and stop the data logger function and store the values in a custom comma separated values (csv) file (see Appendix M).

Figure 25 shows the developed GUI, going from left to right we can see the network control column that allows us to give the network interface id and the initial control method to be used while launching an EtherCAT network with the Start Button. The following column is for joint and cartesian control, while displaying the position of each joint in radians it takes the target values that the user wants to upload with the Send button. Below is cartesian Mode, it applies forward kinematics to the joint positions to find the cartesian position of the end-effector (last joint) of the robot, while taking target positions and processing them with inverse kinematics to move the robot arm. The last column is the data logger section where the program samples data of the position, velocity, and torque of each joint and stores it into a comma separated value (.csv) file. The user interface refreshes automatically the joint and cartesian position values using a dedicated thread.



Type Network Interface: enp3s0

Mode of Operation: CSP

Joint Control

| | Target Position [rad] | Actual Position [rad] | Negative Limit [rad] | Positive Limit [rad] |
|---|---|---|---|---|
| Joint 1 | 0.0 | | -0.0 | 0.0 |
| Joint 2 | 0.0 | | -0.0 | 0.0 |
| Joint 3 | 0.0 | | -0.0 | 0.0 |
| Joint 4 | 0.0 | | -0.0 | 0.0 |
| Joint 5 | 0.0 | | -0.0 | 0.0 |
| Joint 6 | 0.0 | | -0.0 | 0.0 |

Cartesian Mode

| | Target Position [mm] | Actual Position [mm] | Negative Limit [mm] | Positive Limit [mm] |
|---|---|---|---|---|
| dX | 0.0 | | -0.0 | 0.0 |
| dY | 0.0 | | -0.0 | 0.0 |
| dZ | 0.0 | | -0.0 | 0.0 |
| rX | 0.0 | | -0.0 | 0.0 |
| rY | 0.0 | | -0.0 | 0.0 |
| rZ | 0.0 | | -0.0 | 0.0 |

Data Logger (.csv): dataLog.csv

Amount of loops recorded: 0

*Figure 25 Robot Control GUI*

## 6.4 EtherCAT Check Thread

The ecatcheck runs an infinite while loop to guarantee control over the EtherCAT State Machine (see Appendix L). The thread checks if all the slaves are responding and if the setup sequence has triggered the variable inOP. It runs the function ec_readstate() to read the state of each slave state machine. From this point, a for loop runs to check if every slave in the network is in the operational state; if this is not the case, the code will handle the different situations to keep the network operational as they are:

- The slave is in Safe Operation state and has errors -> ack and move to SafeOp

- The slave is in a Safe Operation state -> move to operational
- The slave is in an unknown state - > reconfigure the slave
- The slave is lost -> recover sequence for slave

After this for loop is finished, the thread sleeps for 10 ms and starts again to check the network status.

## 6.5 Real-Time Cycle Aid Functions

To guarantee real-time cycles in the main control thread of the program, some aid functions were used (see Appendix H). Here we can see what they do:

Runner is the real-time thread that runs an infinite while loop; it takes the loop time in microseconds as an argument. It starts by getting the monotonic time of the OS (unmodifiable clock) to define an accurate start time for the thread. Then it sends the EtherCAT Buffer to the network to start the real-time loop. Inside the loop, it begins by calculating how much time is left to start the cycle and waits for it using clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &ts, &tleft). When the variable dorun is TRUE, the thread downloads the EtherCAT Network Buffer with wkc = ec_receive_processdata(EC_TIMEOUTRET) and sets the work counter. Here is where the real-time read/write of the PDOs will happen. From that point, it checks if any slave has a Distributed Clock (DC) clock ec_slave[0].hasdc and ec_sync(ec_DCtime, cycletime, &toff) and adjusts the clock offset in the OS to make the cycle times match the ones on the actuators. Finally, it uploads the EtherCAT Network Buffer and jumps into the next cycle.

## 6.6 Runner Real-Time Thread

The runner code (see Appendix I) uses the SOEM function ec_init(ifname) to call the network interface and start the EtherCAT network; if no connection is made, the code will stop. Then the SOEM function ec_config_init(FALSE) is used to list and initialize all the EtherCAT slaves in the network; the FALSE argument is because the program does not use a configtable. Here the code calls the custom Class Motor and creates an instance for each slave in the network, CubemarsMotor cubemars1((uint16) slave1, 1). From this point, we call the built-in functions of the class to do the following:

- PDO mapping of each slave -> cubemars1.ConfigurePDOs()
- Set all the PDO maps in the network -> CubemarsMotor::ConfigureMotors()
- Change the mode of operation -> cubemars1.ConfigureProfilePositionMode()
- Set RPDO values -> cubemars1.SetTargetPosition((int32_t) 4800)
- Get TPDO values -> cubemars1.GetPositionActualValue()
- Download the EtherCAT Network Buffer -> CubemarsMotor::ReadEthercat()
- Modify the Output Buffer/Datagram -> cubemars1.WritingPDOs()
- Read/decode the Input Buffer/Datagram -> cubemars1.ReadingPDOs()
- Upload the EtherCAT Network Buffer -> CubemarsMotor::WriteEthercat()

With the previous functions, the code can configure and operate the network. Now we use ec_slave[0].state = EC_STATE_INIT to set the EtherCAT State Machine to the initial state, with ec_writestate(0), we send the change to the ecatcheck thread, and thus we change the state of the network. Finally, the function ec_close() stops the network and disconnects it from the interface.

## 6.7 Joystick Control Functions

There are two main functions required to perform the joystick control over the system. The first one is the Joystick Detection function (see Appendix J) as it uses a temporal variable to store the joystick value from the previous iteration (loop) to determine if there is either a raising or a falling edge. After detecting a falling edge from the speed paddle, the code will either increase or decrease the joystick mode of operation (0: X-Y, 1: Z-Roll and 2: Pitch-Yaw). As for the rising edges of the joystick itself the code will update a global variable (command) based on the direction of the joystick. This command variable will be different than 0 until the joystick return to the neutral position.

The second code is the Joystick Send Command (see Appendix K). This function takes the command extracted from the joystick and while it is different than 0 it will maintain a constant increment on the robot actual position based on the inclination of the joystick and the mode of operation. After updating the cartesian position of the robot, the function generates the inverse kinematics for the arm and upload it into a global variable.

## 6.8 Limit Protection Functions

It is security layer on the program to avoid the collision of the system with the environment or the user itself. This function achieves said protection by:

- Comparing stored limit values for position, velocity, and torque with current values.
- In the case of position, both joint and the end effector (coordinate axis), if their target position is within the limits of the system, then the robot will move. As if the joints try to turn beyond the positive or negative limit, they will be restricted to stop or change the

direction (see Appendix N). These limits are manually tuned to avoid the area where both user and power wheelchair exist.

- In the case of velocity and torque if a peak value is read for more than 5 cycles of the program, the robot will be immobilized automatically by changing its EtherCAT state into pre-operational (see Appendix O).
- The user GUI possess extra fields to grab both lower and upper limit of this variables in case the user wants to override the current setup.

## 6.9 SOEM Control Summary

This chapter presented in detail the main functions that compose the SOEM program running in the RTCP. Moreover, it has shown how the functions integrate with one another. Now we present a flowchart that summarizes how this program translates processes and translates signals into the real world.

Figure 26 the Data flowchart of the SOEM code. The figure starts from the left by determining which mode (M: 0,1,2) would be used, as the following column will show how the code handles each operation. For example, the Joystick Control generates a command based on the current Joystick mode; then the program takes the current values of the joints and computes the forward kinematics to find the cartesian position of the end effector. Based on the actual position of the end-effector the joystick command is added to the respective component (linear or rotational X, Y, Z) and then the inverse kinematics is calculated. If the inverse kinematics finds a solution with no singularities the code will proceed to check if both joint and cartesian positions are within the limits of range of operation. If the target values meet the limits criteria the target will be uploaded into the EtherCAT PDOs. After uploading the PDOs the program will take the RTCP internal clock and will request the motors distributed clock to perform and comparison and adjust any possible offset to guarantee the loop time. From that point the program will request the system PDOs back. Based on the returned PDOs the system will check if either the velocity or the torque of the moving motors is crossing the limits, if that happens the operation will be paused for the user to debug the cause of it. Otherwise, it will continue to calculate the inverse kinematics and display the network variables into the GUI.

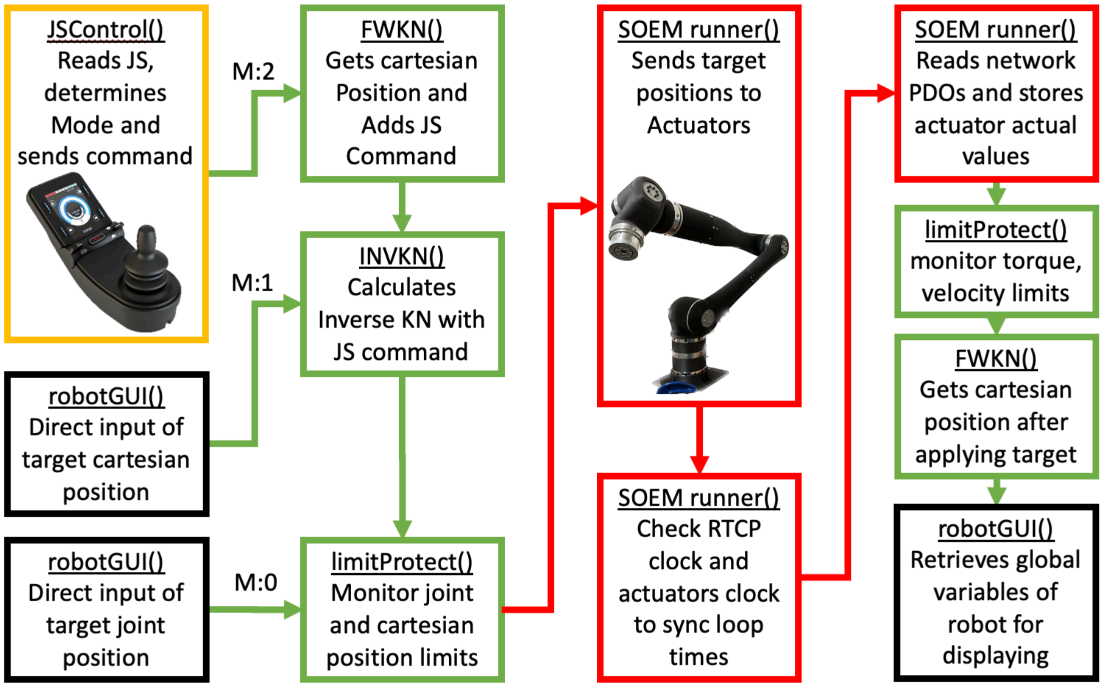

JSControl() Reads JS, determines Mode and sends command
M:2
FWKN() Gets cartesian Position and Adds JS Command
M:1
INVKN() Calculates Inverse KN with JS command
robotGUI() Direct input of target cartesian position
robotGUI() Direct input of target joint position
M:0
limitProtect() Monitor joint and cartesian position limits
SOEM runner() Sends target positions to Actuators
SOEM runner() Check RTCP clock and actuators clock to sync loop times
SOEM runner() Reads network PDOs and stores actuator actual values
limitProtect() monitor torque, velocity limits
FWKN() Gets cartesian position after applying target
robotGUI() Retrieves global variables of robot for displaying

*Figure 26 Data Flow Across SOEM Code*

# Chapter 7: Experiments, Results and Discussions

This chapter presents experiments performed to validate the proposed control system and assistive robotic arm. The first experiments are meant to confirm that the RTCP can run a proper EtherCAT network while keeping a consistent real-time loop, this is vital as an unstable EtherCAT Master will be unable to control the robotic actuators or worse it could trigger unforeseen responses like robot joints crashing over the setup. The next experiments go over the robotic actuator internal control to confirm not only they perform as advertised but that the RTCP can send control signals and retrieve sensor data as it is necessary for the rest of the study. The next experiments built on the top of previous ones as it is meant to test the forward and inverse kinematics to validate that the mathematical models do control the assistive robot arm properly in the real world. The last experiment uses the joystick control based on the flowchart of Figure 23 to validate that the system can be controlled properly and be used in ADLs.

## 7.1 Test of real time cycles using Digital I/O of the actuators

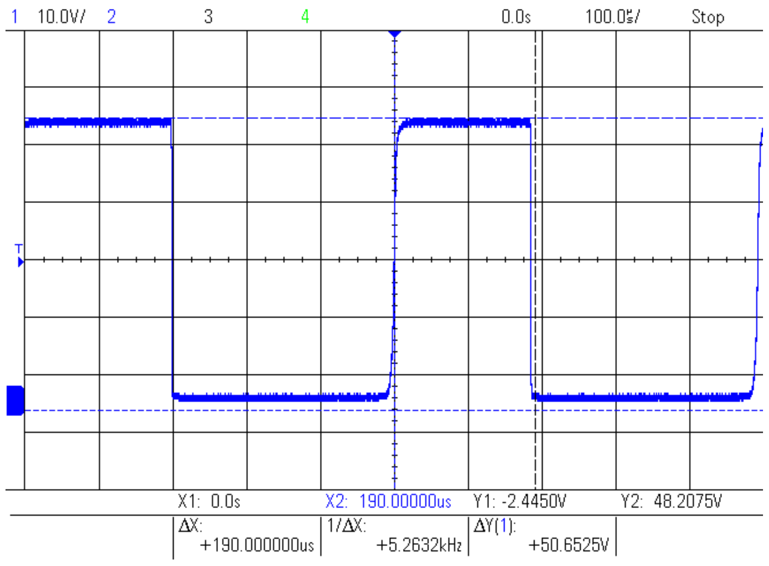


*Figure 27 Oscilloscope Read of Real-Time with Multiple Actuators*

All the EtherCAT network time delay measures shown in the Table 6. are the mean value over 5 measures done in two computing platforms: Jetson Nano, and RT-PC with i7 and Ubuntu 22.04.

*Table 6 EtherCAT Network time delays*

| Slave Number | Measured Delay Time (ns) | | |
|---|---|---|---|
| | Single Act. | Multiple Act. | Robot |
| 1 | 0 | 0 | 0 |
| 2 | 880 | 840 | 820 |
| 3 | - | 1680 | 1660 |
| 4 | - | 2520 | 2500 |
| 5 | - | 3340 | 3320 |
| 6 | - | 4030 | 4160 |
| 7 | - | 4960 | 5020 |

The results seen on Table 7 are measures of the program's loop time. The measures were taken using a Keysight Oscilloscope for the multiple configurations of the experimental setup.

*Table 7 Real-Time Thread Cycle Time Measures*

| Target Cycle Time (us) | Measured Cycle Time (us) | | |
|---|---|---|---|
| | Single Act. | Multiple Act. | Robot |
| 300 | 260 | 250 | 256 |
| 400 | 360 | 350 | 354 |

|  |  |  |  |
|---|---|---|---|
| 500 | 450 | 460 | 460 |
| 600 | 560 | 550 | 570 |
| 700 | 660 | 660 | 650 |
| 800 | 750 | 760 | 750 |
| 900 | 860 | 870 | 872 |
| 1000 | 980 | 960 | 953 |
| 2000 | 1940 | 1920 | 1922 |

Table 7 contains the results of the tests. The oscilloscope measurements revealed a pattern. Figures 27 and 28 show the Target Cycle Time result from a Multiple Actuator experiment test configuration, yet they diverge on the falling edge time making the Cycle Time be in a consistent off set of 25 to 50 microseconds. In other words. This showed the need for a more accurate calculation of the loop times.

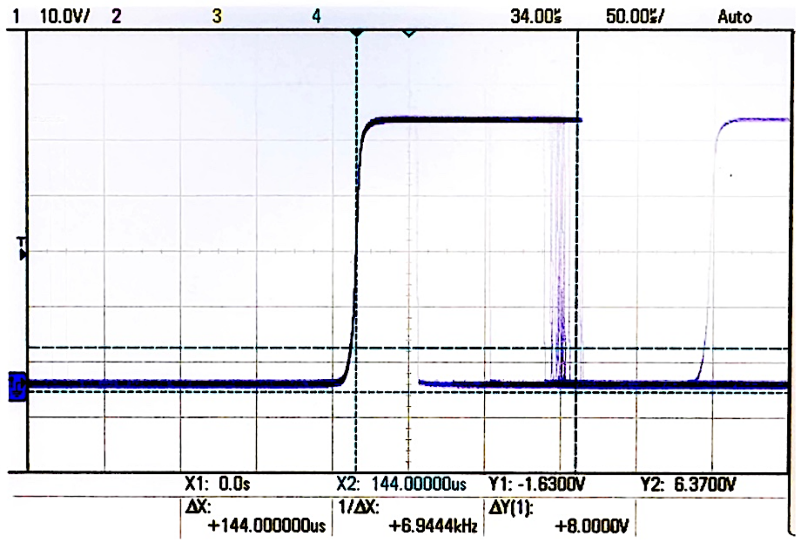


*Figure 28 Oscilloscope capture fluctuation*

Using the functionality of the oscilloscope to export comma separated values (csv) files, a test run was made to gather some files further analysis.

Based on the readings we can compute the overall Accuracy and Precision of the System with respect to Real-Time Cycles with a target of $200\ \mu s$.

$$Average\ value = \frac{sum\ of data}{number\ of\ measuraments} \tag{16}$$

$$Percent\ error = \frac{target\ value - measured\ value}{target\ value} * 100 \tag{17}$$

$$Absolute\ deviation = measured\ value - average\ value \tag{18}$$

$$Average\ deviation = \frac{Sum\ of\ absolute\ deviations}{number\ of\ measurements} \tag{19}$$

Using the previous equations, a Matlab code was developed to open and process the csv files and extract the statistics quickly. From a capture of a thousand samples, it was found that the average deviation is: $218.1\ \mu s \pm 21.4\ \mu s$ with a percent error of 9%.

More experimental data was recorded, and the following table shows its results.

*Table 8 Real-Time Thread Cycle Stats*

| Target Cycle Time (us) | Average Value (us) | Average Deviation (us) | Percent Error (%) |
|---|---|---|---|
| 300 | 274 | 23 | 8.6 |
| 400 | 372 | 39 | 7 |
| 500 | 463 | 46 | 7.4 |
| 600 | 558 | 67 | 7 |
| 700 | 654 | 40 | 6.57 |
| 800 | 758 | 42 | 6.5 |
| 900 | 843 | 56 | 6.33 |
| 1000 | 940 | 60 | 6 |
| 2000 | 1879 | 120 | 6.04 |

One possible reason for the accuracy of this system could be related to the electrical components of the EtherCAT IO Box since it has optocouplers for protection, that added layer to activate and deactivate the Outputs of the system can introduce potential delays as it would be for this case. Another possible reason is the fact that the study uses soft real-time because of how the shared resources are used over the code, allowing to have small offsets in the timing of the cycles. Following experiments should consider using a different EtherCAT IO slave and a different real-time implementation like Xenomai to confirm or deny this.

## 7.2 Tests of Arm Cyclic Synchronous Position

The following tests were performed to the robotic actuators to validate their capabilities and check that computational, electrical, and mechanical components are performing as expected. The following figure shows the readings of cyclic synchronous position control applied to the robotic actuator. The reference value is a square signal with a magnitude of 1.35 radians. The velocity limit is of 0.2 radians per second

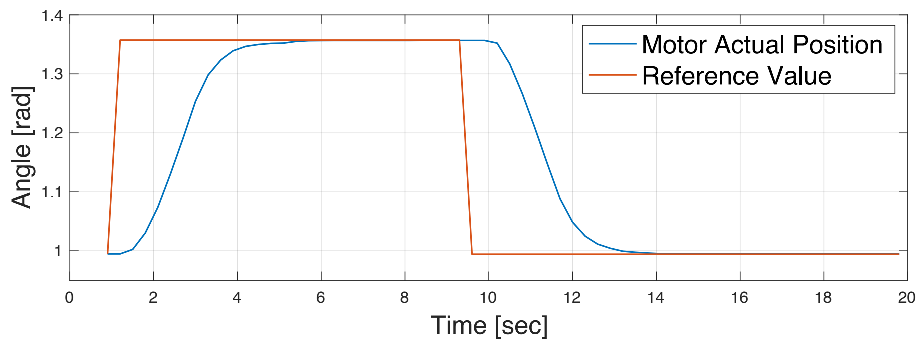


*Figure 29 Angular position of single robotic actuator with CSP*

From Figure 29 the motor’s position has a settling time of 6 seconds and a steady state error 0.0007 radians which is under the precision mentioned in (Lyy et al., 2022).

## 7.3 Tests of Arm Inverse Kinematics

The following tests were performed to the assistive robotic arm to determine its capabilities and check that computational, electrical, and mechanical components are performing as expected. The following Figure 30 shows the readings of inverse kinematics control applied to the robot arm.

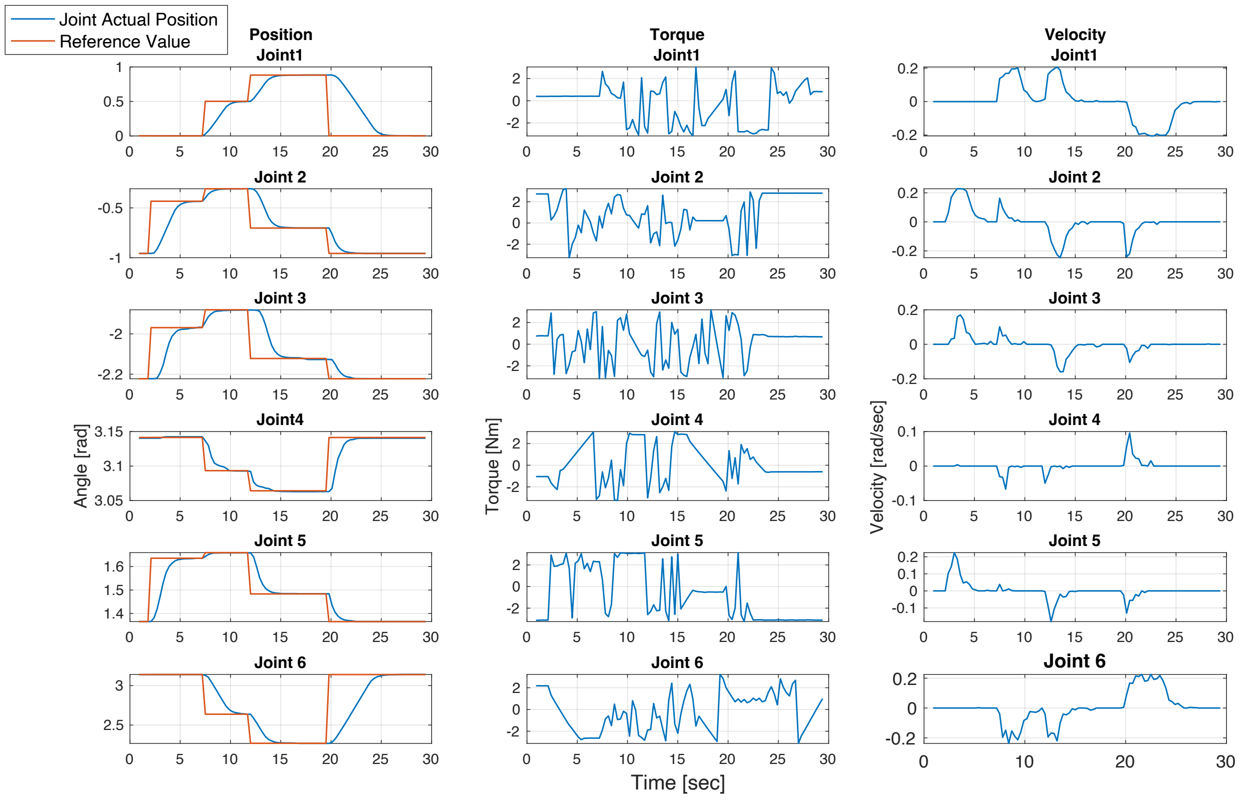


*Figure 30 Angular position, torque, and speed of 200 by 200 mm box along the X-Y axis*

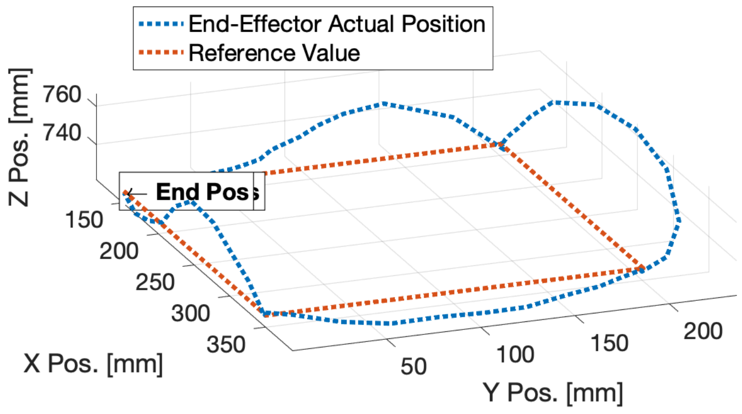


*Figure 31 Cartesian trajectory of a 200 by 200 mm box along the X-Y axis*

From the recorded data of the assistive arm, the joint angles, torque, and velocity were extracted and shown in Figure 31 for tracing a square over the X-Y axis and Figure 33 for tracing a square over the Y-Z axis. Both figures have some differences, as the elapsed time is 33% less to draw the square in the Y-Z axis; as for the torques, the last 5 seconds of each trajectory shows how the inverse kinematics assigns more rotations on the joint 1 to do a displacement in the Y axis while the joint 3 has more rotations when a displacement along the Z axis is performed. Moreover, the trajectory of the square in the Y-Z axis shows an increased torque in the joint 2 and 3 since that joint is holding the weight of the following joints and perform most of the actuation as the robot get further away from its origin axis.

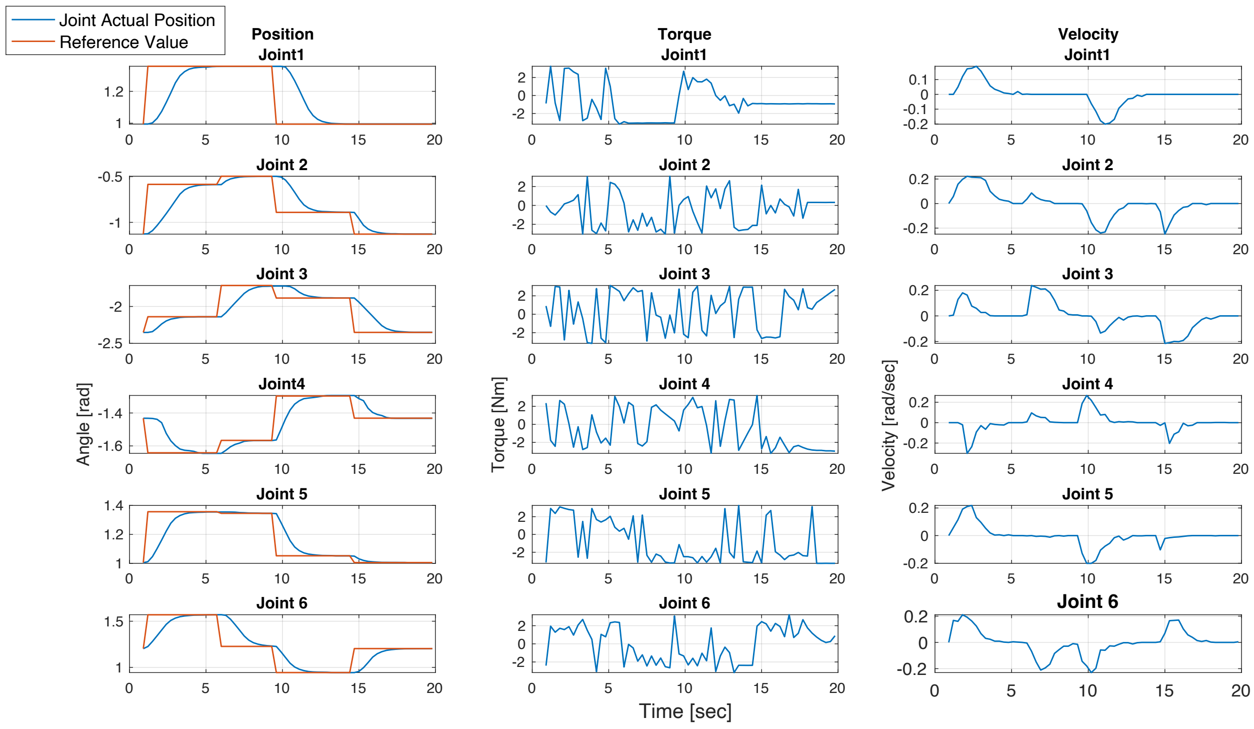


*Figure 32 Angular position, torque, and speed of 200 by 200 mm box along the Y-Z axis*

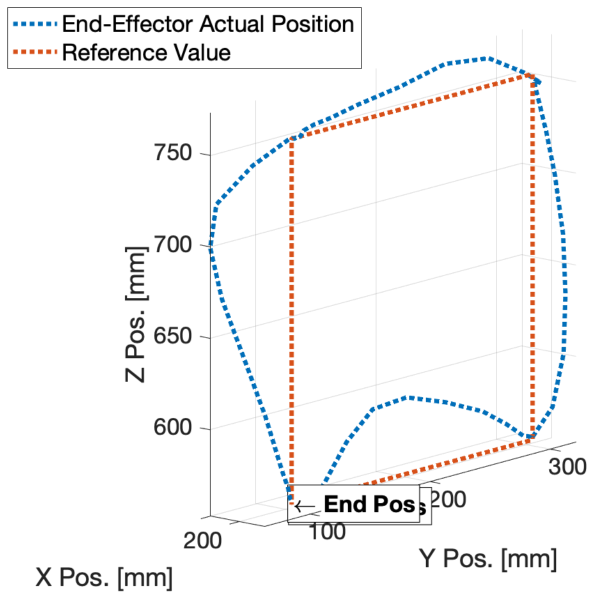


*Figure 33 Cartesian trajectory of a 200 by 200 mm box along the Y-Z axis*

Based on the shown results it was clear that we could move forward to the experiments using the joystick module as it will work applying inverse kinematics over small steps that will be updated cyclically.

## 7.4 Tests of Joystick Control

This research presents pick and place tasks from three different locations: ground, table, and shelf. Two studies were approved by the UWM IRB (21.306. UWM and 21.309. UWM) to determine the performance of the implemented finger joystick and chin joystick system in a previous publication (Rulik et al., 2022).

Figure 21 shows the final setup of the system. The power wheelchair with the assistive robotic arm is mounted on the left side rail. Two user inputs control the robotic arm and the powered wheelchair. Joystick control is done with a two-axis joystick. There is a toggle switch to change the control between the wheelchair and the Assistive Arm.

For safety measures, the robot arm's active workspace is reduced to avoid any contact with the user. As daily living tasks, picking and placing items from the ground, a table and shelf were selected due to their change in height. The data from the robotic arm was gathered for the joystick experiments allowing to display of the trajectory of the robot end-effector. In the ADL tasks the highest item was 0.8 m above the ground, and the furthest away object horizontally was 0.692 m. In these tasks, the Cartesian mode allows manipulating the robotic arm based on the theory seen in Chapter 4 and 5 by updating the target position using the joysticks. Figure 35a shows the trajectory followed by the robot gripper for the joystick to pick an object from the table. The dotted blue line is the trajectory using the joystick, and the dotted red line represents the target position sent. Likewise, Figure 35c shows the end-effector path to picking an object from the ground. And 8c the trajectory to pick and object from a shelf.

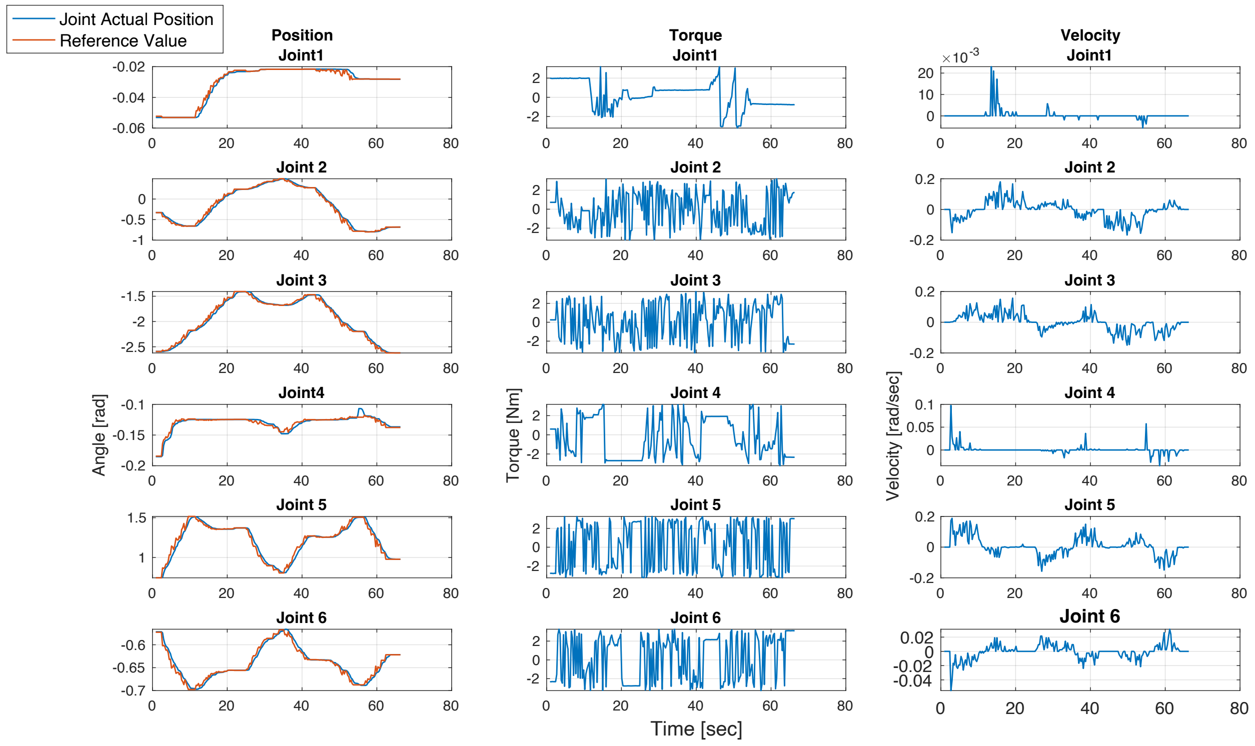


*Figure 34 Angular position, torque, and speed of pick form table ADL*

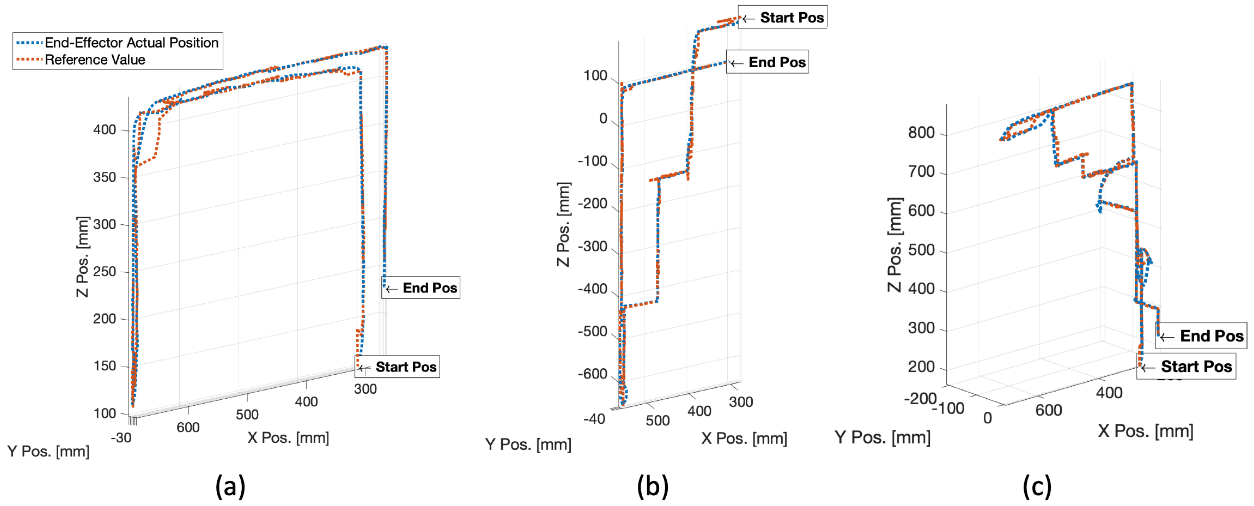
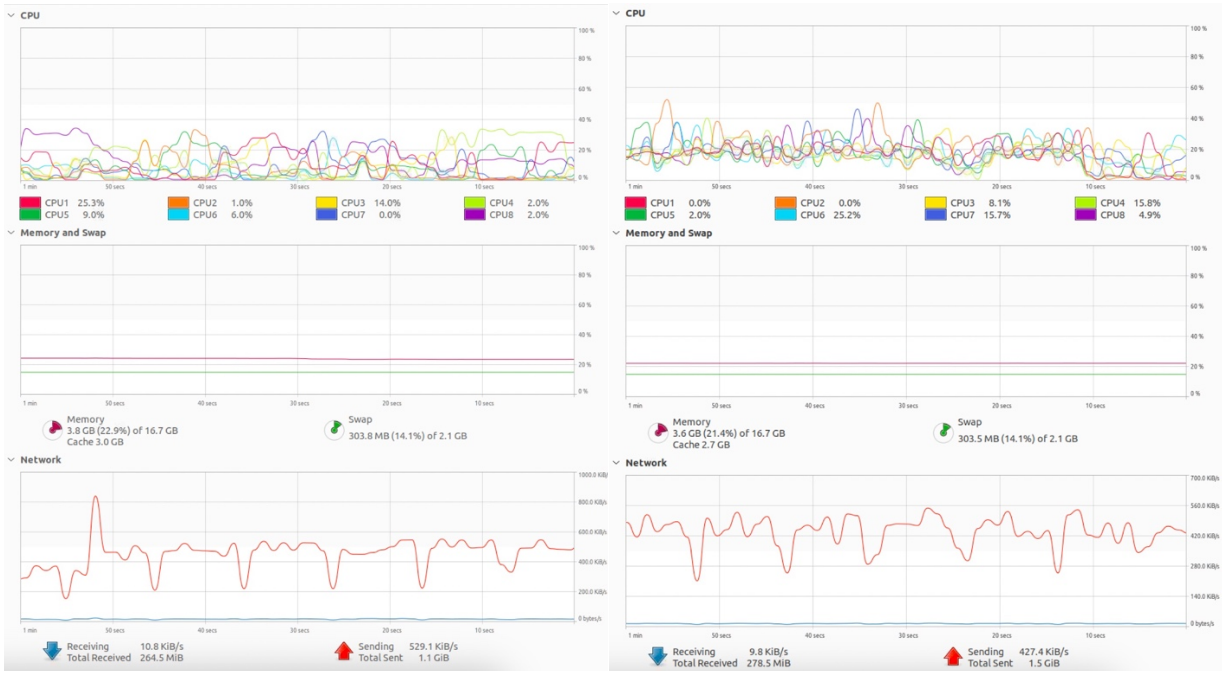


*Figure 35 Cartesian trajectory of picking an item from (a) table, (b) ground and (c) shelf*

## 7.5 Tests of Computing Resource Utilization

After performing an adequate control over the 6DOF robot arm and power wheelchair system. Measuring and comparing the computing resource utilization of the RTCP, the jetson nano, its desired since it can demonstrate the capabilities of the control platform to perform over long periods of time. As a reference we used a Dell OptiPlex computer paired with an Intel i7 8th generation and 16 gb of RAM to run the software simultaneously.



```
CPU1   25.3%     CPU2   1.0%      CPU3   14.0%     CPU4   2.0%
CPU5    9.0%     CPU6   6.0%      CPU7    0.0%     CPU8   2.0%

Memory
3.8 GB (22.9%) of 16.7 GB          Swap
Cache 3.0 GB                       303.8 MB (14.1%) of 2.1 GB

Receiving        10.8 KiB/s        Sending      529.1 KiB/s
Total Received   264.5 MiB         Total Sent   1.1 GiB
```

```
CPU1    0.0%     CPU2   0.0%      CPU3    8.1%     CPU4   15.8%
CPU5    2.0%     CPU6   25.2%     CPU7   15.7%     CPU8    4.9%

Memory
3.6 GB (21.4%) of 16.7 GB          Swap
Cache 2.7 GB                       303.5 MB (14.1%) of 2.1 GB

Receiving        9.8 KiB/s         Sending      427.4 KiB/s
Total Received   278.5 MiB         Total Sent   1.5 GiB
```

*Figure 36 Resource Utilization of Reference Platform, before running the program (left) and while running (right)*

On the other hand, the jetson nano showed the results found in figure 35. The performance of the jetson nano does take more CPU resources to run the program.

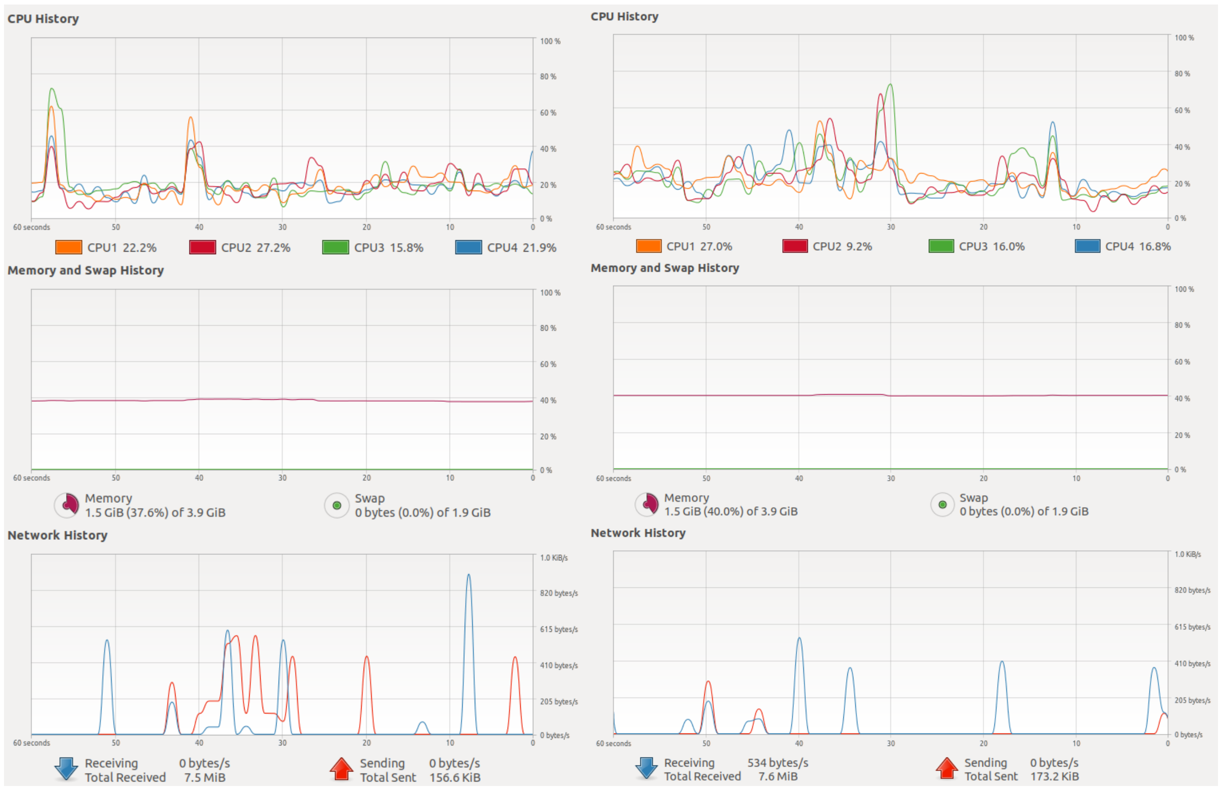


*Figure 37 Resource Utilization of RTCP Platform, before running the program (left) and while running (right)*

Form comparing figures 36 and 37 we can see that there is not a considerable difference between the system resources utilization since: average CPU usage before running the program is around 10% and when running the program, it is 20%. The memory and network components are not displaying a change worth to be mentioned. As for the jetson nano its average CPU usage is around 20% before running the program and then it goes around 30% while running the program even thought that it presents spikes in its CPU core utilization.

From this experiment we can conclude that the Dell OptiPlex platform performs better, but more over we can conclude that the jetson nano platform performs adequately to hold the whole load of the program as it is using less than 30% of its available resources. A point to be made is that while performing this experiment the jetson nano presented more fluctuations on the resource utilization window while browsing the file system and launching the screen capture programs.

# Chapter 8: Conclusions and Future works

This study aimed to develop a Real Time Control Platform (RTCP) and integrate it with a multimodal control system to assist individuals with restricted mobility in performing ADL while having control over their mobility and environment. The wheelchair and a wheelchair-mounted robotic arm are considered, combined, and then translated the control commands to control manually in Cartesian mode to accomplish this task. The proposed control system was put to the test with computed trajectories using the GUI and manual trajectories given by the joystick. The analytical solution of the inverse kinematics proved to be useful to control the robotic arm. For safety measures, the assistive robot arm workspace was limited to avoid contact with the user. Experiments results guarantee that the proposed control framework can suitably be used to maneuver both a powered wheelchair and an assistive robot for ADL assistance using the integrated wheelchair joysticks. This research thus contributes to significant technological advancement that has yielded a novel portable control platform with the implementation of a control framework that uses a joystick. Note that the developed control framework requires validation with healthy participants with a successful completion of ADL tasks as the perquisite before testing the system with individuals with upper extremity dysfunctions. The comparison between the joystick and reference operations in ADLs is also demonstrated. The distribution of the task completion time indicates that the joystick operation was more time-consuming than computed trajectory.

The following steps for this research include expanding the user input devices, perform experiments with healthy individuals and add a wider range of ADLs to test. All to conduct future experiments with wheelchair users having limited or no upper limb movements.

# References

Alizadeh, A., Dyck, S. M., & Karimi-Abdolrezaee, S. (2019). Traumatic spinal cord injury: an overview of pathophysiology, models and acute injury mechanisms. *Frontiers in Neurology*, *10*, 282.

Alonso-Martin, F., & Salichs, M. A. (2011). Integration of a voice recognition system in a social robot. *Cybernetics and Systems: An International Journal*, *42*(4), 215–245.

Andreasen Struijk, L. N. S., Egsgaard, L. L., Lontis, R., Gaihede, M., & Bentsen, B. (2017). Wireless intraoral tongue control of an assistive robotic arm for individuals with tetraplegia. *Journal of Neuroengineering and Rehabilitation*, *14*(1), 1–8.

Ansari, M. F., Edla, D. R., Dodia, S., & Kuppili, V. (2019). Brain-computer interface for wheelchair control operations: An approach based on fast fourier transform and on-line sequential extreme learning machine. *Clinical Epidemiology and Global Health*, *7*(3), 274–278.

Baldi, T. L., Spagnoletti, G., Dragusanu, M., & Prattichizzo, D. (2017). Design of a wearable interface for lightweight robotic arm for people with mobility impairments. *2017 International Conference on Rehabilitation Robotics (ICORR)*, 1567–1573.

Brinkman, A., Morris, J., Chen, I., Sheikh, N., & Warren, P. (2021). Fastcat: An Open-Source Library for Composable EtherCAT Control Systems. *2021 IEEE Aerospace Conference (50100)*, 1–8. https://doi.org/10.1109/AERO50100.2021.9438315

Chuang, W.-L., Yeh, M.-H., & Yeh, Y.-L. (2021). Develop Real-Time Robot Control Architecture Using Robot Operating System and EtherCAT. *Actuators*, *10*(7). https://doi.org/10.3390/act10070141

Cong, V. D., Hanh, L. D., Phuong, L. H., & Duy, D. A. (2022). Design and development of robot arm system for classification and sorting using machine vision. *FME Transactions*, *50*(1), 181.

Cowan, R. E., Fregly, B. J., Boninger, M. L., Chan, L., Rodgers, M. M., & Reinkensmeyer, D. J. (2012). Recent trends in assistive technology for mobility. *Journal of Neuroengineering and Rehabilitation*, *9*(1), 1–8.

Craig, T. L., Nelson, C. A., Li, S., & Zhang, X. (2016). Human gaze commands classification: A shape based approach to interfacing with robots. *2016 12th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA)*, 1–6.

Curtiss-Wright. (2022). *IOM User Manual*.

Curtiss. (2022). *PG Drives Technology. R-Net Technical Manual SK77981/14*.

Delgado, R., & Choi, B. W. (2019). Network-Oriented Real-Time Embedded System Considering Synchronous Joint Space Motion for an Omnidirectional Mobile Robot. *Electronics*, *8*(3). https://doi.org/10.3390/electronics8030317

Delgado, R., Kim, S.-Y., You, B.-J., & Choi, B.-W. (2016). An EtherCAT-based real-time motion control system in mobile robot application. *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 710–715. https://doi.org/10.1109/URAI.2016.7734098

Delgado, R., You, B.-J., & Choi, B. W. (2019). Real-time control architecture based on Xenomai using ROS packages for a service robot. *Journal of Systems and Software*, *151*, 8–19. https://doi.org/https://doi.org/10.1016/j.jss.2019.01.052

Fall, C. L., Quevillon, F., Blouin, M., Latour, S., Campeau-Lecours, A., Gosselin, C., & Gosselin, B. (2018). A multimodal adaptive wireless control interface for people with upper-body disabilities. *IEEE Transactions on Biomedical Circuits and Systems*, *12*(3), 564–575.

Fall, C. L., Turgeon, P., Campeau-Lecours, A., Maheu, V., Boukadoum, M., Roy, S., … Gosselin, B. (2015). Intuitive wireless control of a robotic arm for people living with an upper body disability. *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 4399–4402.

Francis, W. C., Umayal, C., & Kanimozhi, G. (2021). Brain-Computer Interfacing for Wheelchair Control by Detecting Voluntary Eye Blinks. *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, *9*(2), 521–537.

Fu, Z., Yang, W., & Yang, Z. (2013). Solution of inverse kinematics for 6R robot manipulators with offset wrist based on geometric algebra. *Journal of Mechanisms and Robotics*, *5*(3), 31010.

Hildebrand, M., Bonde, F., Kobborg, R. V. N., Andersen, C., Norman, A. F., Thøgersen, M., … Struijk, L. N. S. A. (2019). Semi-autonomous tongue control of an assistive robotic arm for individuals with quadriplegia. *2019 IEEE 16th International Conference on Rehabilitation Robotics (ICORR)*, 157–162.

Hu, F., Qu, F., Li, J., & Zhao, H. (2022). Real-time research based on ARM-based robot EtherCAT master system. *2022 International Symposium on Control Engineering and Robotics (ISCER)*, 12–19. https://doi.org/10.1109/ISCER55570.2022.00009

Jansen, D., & Buttner, H. (2004). Real-time Ethernet: the EtherCAT solution. *Computing and Control Engineering*, *15*(1), 16–21.

Jiang, H., Wachs, J. P., Pendergast, M., & Duerstock, B. S. (2013). 3D joystick for robotic arm control by individuals with high level spinal cord injuries. *2013 IEEE 13th International Conference on Rehabilitation Robotics (ICORR)*, 1–5.

Kalgaonkar, K., & Raj, B. (2009). One-handed gesture recognition using ultrasonic Doppler sonar. *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, 1889–1892.

Kang, S. J. (2021). A Study on Implementation of Real-time EtherCAT Master. *Journal of the Semiconductor \& Display Technology*, *20*(2), 131–136.

Klebbe, R., Scherzinger, S., Eicher, C., & others. (2022). Assistive Robots for Patients With Amyotrophic Lateral Sclerosis: Exploratory Task-Based Evaluation Study With an Early-Stage Demonstrator. *JMIR Rehabilitation and Assistive Technologies*, *9*(3), e35304.

Langlois, K., van der Hoeven, T., Rodriguez Cianca, D., Verstraten, T., Bacek, T., Convens, B., … Vanderborght, B. (2018). EtherCAT Tutorial: An Introduction for Real-Time Hardware Communication on Windows [Tutorial]. *IEEE Robotics & Automation Magazine*, *25*(1), 22–122. https://doi.org/10.1109/MRA.2017.2787224

Lee, K., Lee, J., Woo, B., Lee, J., Lee, Y.-J., & Ra, S. (2018). Modeling and Control of a Articulated Robot Arm with Embedded Joint Actuators. *2018 International Conference on Information and Communication Technology Robotics (ICT-ROBOT)*, 1–4. https://doi.org/10.1109/ICT-ROBOT.2018.8549903

Lee, Y., Lee, W., Choi, B., Park, G., & Park, Y. (2016). Reliable software architecture design with EtherCAT for a rescue robot. *2016 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*, 34–39. https://doi.org/10.1109/IRIS.2016.8066062

Li, J., Yu, H., Shen, N., Zhong, Z., Lu, Y., & Fan, J. (2021). A novel inverse kinematics method for 6-DOF robots with non-spherical wrist. *Mechanism and Machine Theory*, *157*, 104180.

Lyy, S., Zcx, C., Twy, K., Cwk, K., & Jwj, V. (2022). *RR-HA Series Servo Driver User guide V0.1.6*. Retrieved from http://www.motorevo.cn/index.html

Malkin, J., Li, X., Harada, S., Landay, J., & Bilmes, J. (2011). The vocal joystick engine v1. 0. *Computer Speech & Language*, *25*(3), 535–555.

McKee, A. C., & Daneshvar, D. H. (2015). The neuropathology of traumatic brain injury. *Handbook of Clinical Neurology*, *127*, 45–66.

Memon, Y. A., Motan, I., Akbar, M. A., Hameed, S., & Hasan, M. U. (2016). Speech recognition system for a voice controlled robot with real time obstacle detection and avoidance. *International Journal Of Electrical, Electronics And Data Communication*, *4*(9), 33–37.

Minetto, M. A., Giannini, A., McConnell, R., Busso, C., Torre, G., & Massazza, G. (2020). Common musculoskeletal disorders in the elderly: the star triad. *Journal of Clinical Medicine*, *9*(4), 1216.

Mlinac, M. E., & Feng, M. C. (2016). Assessment of activities of daily living, self-care, and independence. *Archives of Clinical Neuropsychology*, *31*(6), 506–516.

Nishimori, M., Saitoh, T., & Konishi, R. (2007). Voice controlled intelligent wheelchair. *SICE Annual Conference 2007*, 336–340.

Paprocki, M., & Erwiński, K. (2022). Synchronization of Electrical Drives via EtherCAT Fieldbus Communication Modules. *Energies*, *15*(2), 604.

Păsărică, A., Bozomitu, R. G., Cehan, V., Lupu, R. G., & Rotariu, C. (2015). Pupil detection algorithms for eye tracking applications. *2015 IEEE 21st International Symposium for Design and Technology in Electronic Packaging (SIITME)*, 161–164.

Penaloza, C. I., & Nishio, S. (2018). BMI control of a third arm for multitasking. *Science Robotics*, *3*(20), eaat1228.

Penkert, H., Baron, J. C., Madaus, K., Huber, W., & Berthele, A. (2021). Assessment of a novel, smartglass-based control device for electrically powered wheelchairs. *Disability and*

*Rehabilitation: Assistive Technology*, *16*(2), 172–176.

Permobil. (2022a). *Compact Joystick*.

Permobil. (2022b). *Joystick Module w/Bluetooth*.

Permobil. (2022c). *M3 Corpus*.

Perrin, S., Cassinelli, A., & Ishikawa, M. (2004). Gesture recognition using laser-based tracking system. *Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings.*, 541–546.

Pfeiffer, O., Ayre, A., & Keydel, C. (2008). *Embedded networking with CAN and CANopen*. Copperhill Media.

Puanhvuan, D., Khemmachotikun, S., Wechakarn, P., Wijarn, B., & Wongsawat, Y. (2017). Navigation-synchronized multimodal control wheelchair from brain to alternative assistive technologies for persons with severe disabilities. *Cognitive Neurodynamics*, *11*(2), 117–134.

Pulikottil, T. B., Caimmi, M., D'Angelo, M. G., Biffi, E., Pellegrinelli, S., & Tosatti, L. M. (2018). A voice control system for assistive robotic arms: preliminary usability tests on patients. *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob)*, 167–172.

Rabhi, Y, Mrabet, M., & Fnaiech, F. (2015). Optimized joystick control interface for electric powered wheelchairs. *2015 16th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, 201–206.

Rabhi, Yassine, Mrabet, M., Fnaiech, F., & Gorce, P. (2013). Intelligent joystick for controlling power wheelchair navigation. *3rd International Conference on Systems and Control*, 1020–1025.

Reghenzani, F., Massari, G., & Fornaciari, W. (2019). The Real-Time Linux Kernel: A Survey on PREEMPT_RT. *ACM Comput. Surv.*, *52*(1). https://doi.org/10.1145/3297714

Ren, Z., Yuan, J., Meng, J., & Zhang, Z. (2013). Robust part-based hand gesture recognition using kinect sensor. *IEEE Transactions on Multimedia*, *15*(5), 1110–1120.

Rudigkeit, N., & Gebhard, M. (2019). AMiCUS—A Head Motion-Based Interface for Control of an Assistive Robot. *Sensors*, *19*(12). https://doi.org/10.3390/s19122836

Rulik, I., Sunny, M. S. H., De Caro, J. D. S., Zarif, M. I. I., Brahmi, B., Ahamed, S. I., … others. (2022). Control of a wheelchair-mounted 6DOF assistive robot with Chin and finger joysticks. *Frontiers in Robotics and AI*, *9*.

Siciliano, B., Khatib, O., & Kröger, T. (2008). *Springer handbook of robotics* (Vol. 200). Springer.

Solea, R., Margarit, A., Cernega, D., & Serbencu, A. (2019). Head movement control of powered wheelchair. *2019 23rd International Conference on System Theory, Control and Computing (ICSTCC)*, 632–637.

Starke, S., Hendrich, N., & Zhang, J. (2018). Memetic evolution for generic full-body inverse kinematics in robotics and animation. *IEEE Transactions on Evolutionary Computation*, *23*(3), 406–420.

Sygulla, F., Wittmann, R., Seiwald, P., Berninger, T., Hildebrandt, A., Wahrmann, D., & Rixen, D. (2018). An EtherCAT-Based Real-Time Control System Architecture for Humanoid Robots. *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, 483–490. https://doi.org/10.1109/COASE.2018.8560532

Tangcharoensathien, V., Witthayapipopsakul, W., Viriyathorn, S., Patcharanarumol, W., & others. (2018). Improving access to assistive technologies: challenges and solutions in low-

and middle-income countries. *WHO South-East Asia Journal of Public Health*, *7*(2), 84.

Taylor, D. M. (2018). Americans with disabilities: 2014. *US Census Bureau*, 1–32.

Thorp, E. B., Abdollahi, F., Chen, D., Farshchiansadegh, A., Lee, M.-H., Pedersen, J. P., … Mussa-Ivaldi, F. A. (2015). Upper body-based power wheelchair control interface for individuals with tetraplegia. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, *24*(2), 249–260.

Tolani, D., Goswami, A., & Badler, N. I. (2000). Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models*, *62*(5), 353–388.

Toro-Hernández, M. L., Kankipati, P., Goldberg, M., Contepomi, S., Tsukimoto, D. R., & Bray, N. (2019). Appropriate assistive technology for developing countries. *Physical Medicine and Rehabilitation Clinics*, *30*(4), 847–865.

Try, P., Schöllmann, S., Wöhle, L., & Gebhard, M. (2021). Visual Sensor Fusion Based Autonomous Robotic System for Assistive Drinking. *Sensors*, *21*(16). https://doi.org/10.3390/s21165419

Tsai, C.-C., Tai, F.-C., Lin, C.-A., & Chan, C.-C. (2019). EtherCAT-based Impedance Control of a 6-DOF Industrial Robotic Manipulator. *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 80–85. https://doi.org/10.1109/AIM.2019.8868411

Valk, T. A., Mouton, L. J., Otten, E., & Bongers, R. M. (2019). Fixed muscle synergies and their potential to improve the intuitive control of myoelectric assistive technology for upper extremities. *Journal of Neuroengineering and Rehabilitation*, *16*(1), 1–20.

Wang, S., Yang, X., & Geer, J. van der. (2021). Development of EtherCAT real-time control system for robot based on Simulink Real-Time. *Journal of Computational Methods in*

*Sciences and Engineering*, *21*, 49–57. https://doi.org/10.3233/JCM-204325

Xin, S. Z., Feng, L. Y., Bing, H. L., & Li, Y. T. (2007). *A simple method for inverse kinematic analysis of the general 6R serial robot*.

Yiyang, L., Xi, J., Hongfei, B., Zhining, W., & Liangliang, S. (2021). A general robot inverse kinematics solution method based on improved PSO algorithm. *IEEE Access*, *9*, 32341–32350.

Zhang, G., Ni, F., Li, Z., & Liu, H. (2018). A Control System Design for 7-DoF Light-weight Robot based on EtherCAT Bus. *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2169–2174. https://doi.org/10.1109/ICMA.2018.8484317

Zhang, J., Zhang, W., & Zhang, M. (2021). Solvable Criteria and Sequential Solution Method for the General 6R Inverse Kinematics Problem. *2021 IEEE International Conference on Mechatronics and Automation (ICMA)*, 17–21.

Zhao, R., Shi, Z., Guan, Y., Shao, Z., Zhang, Q., & Wang, G. (2018). Inverse kinematic solution of 6R robot manipulators based on screw theory and the Paden--Kahan subproblem. *International Journal of Advanced Robotic Systems*, *15*(6), 1729881418818297.

Zhou, S., Fei, F., Zhang, G., Mai, J. D., Liu, Y., Liou, J. Y. J., & Li, W. J. (2013). 2D human gesture tracking and recognition by the fusion of MEMS inertial and vision sensors. *IEEE Sensors Journal*, *14*(4), 1160–1170.

# Appendix A: Main Function

```
int main (int argc, char *argv[])
{
   Main kit(argc, argv);
   GTKRobot GTKRobot;
   Main::run(GTKRobot);
   return 0;
   printf("non reach");
}
```

# Appendix B: GUI Refresh Thread

```
void backgroundRefresh()
   {
   while(STOP_ECAT==FALSE)
     {
      std::string pos1 = std::to_string(theta[0]);
      std::string pos2 = std::to_string(theta[1]);
      std::string pos3 = std::to_string(theta[2]);
      std::string pos4 = std::to_string(theta[3]);
      std::string pos5 = std::to_string(theta[4]);
      std::string pos6 = std::to_string(theta[5]);
      Robot.ForwardKinematics( theta, FW );
      std::string dx = std::to_string(FW[0]);
      std::string dy = std::to_string(FW[1]);
      std::string dz = std::to_string(FW[2]);
      std::string rx = std::to_string(FW[3]);
      std::string ry = std::to_string(FW[4]);
      std::string rz = std::to_string(FW[5]);
      entryActualValue1.set_text(pos1);
      entryActualValue2.set_text(pos2);
      entryActualValue3.set_text(pos3);
      entryActualValue4.set_text(pos4);
      entryActualValue5.set_text(pos5);
      entryActualValue6.set_text(pos6);
      entryActualDX.set_text(dx);
      entryActualDY.set_text(dy);
      entryActualDZ.set_text(dz);
      entryActualRX.set_text(rx);
      entryActualRY.set_text(ry);
      entryActualRZ.set_text(rz);
      if(checkButtonDataLogger.get_active())
      {
         recordData = TRUE;
         logFileName = entryDataLogger.get_text();
         std::string logCount = std::to_string(logCounter);
         entryDataLoggerCurrent.set_text(logCount);
      }
      else
      {
         recordData = FALSE;
         entryDataLoggerCurrent.set_text("0");
      }
      ik_sent = FALSE;
      sleep(1);
      cout.flush();
    }
  }
```

# Appendix C: GUI Start and Stop Network Buttons

```
protected: void buttonStartETCClicked()
   {
      string networkI = entryNetworkInterface.get_text();
      ifname= networkI.c_str(); // IMPORTANT
      string modeOfOperation = comboBoxModeofOp.get_active_text();
      if (modeOfOperation == "CSP")
      {
         modeGUI = 1;
      }
      else if (modeOfOperation == "Cartesian")
      {
         modeGUI = 2;
      }
      else if (modeOfOperation == "Joystick")
      {
         modeGUI = 3;
      }
      else
      {
         throw runtime_error("Unrecognized mode of operation!");
      }
      int ctime = 5000; // cycletime in microseconds
      STOP_ECAT = FALSE;
      osal_thread_create(&thread2, 128000, &ecatcheck, NULL);
      osal_thread_create_rt(&thread1, 128000, &runner, (void*) &ctime);
      Glib::Thread::create(sigc::mem_fun(*this, &GTKRobot::backgroundRefresh), true);
      printf("EtherCAT Network Launched");
   }

protected: void buttonStopETCClicked()
   {
      STOP_ECAT = TRUE;
      printf("EtherCAT Network Stop\n");
   }
```

## Appendix D: GUI Mode of Operation Button

```
protected: void buttonModeofOpClicked()
   {
      string modeOfOperation = comboBoxModeofOp.get_active_text();
      if (modeOfOperation == "CSP")
      {
         modeGUI = 1;
      }
      else if (modeOfOperation == "Cartesian")
      {
         modeGUI = 2;
      }
      else if (modeOfOperation == "Joystick")
      {
         modeGUI = 3;
         jsMode = 0;
      }
      else
      {
         throw runtime_error("Unrecognized mode of operation!");
      }
   }
```

# Appendix E: GUI Send Joint and Cartesian Target Position Buttons

```
protected: void buttonSendTargetClicked()
   {
      string target1 = entryTargetValue1.get_text();
      string target2 = entryTargetValue2.get_text();
      string target3 = entryTargetValue3.get_text();
      string target4 = entryTargetValue4.get_text();
      string target5 = entryTargetValue5.get_text();
      string target6 = entryTargetValue6.get_text();
      targetJointVal[0] = stod(target1);
      targetJointVal[1] = stod(target2);
      targetJointVal[2] = stod(target3);
      targetJointVal[3] = stod(target4);
      targetJointVal[4] = stod(target5);
      targetJointVal[5] = stod(target6);
      robot_targetJointVal[0] = int(targetJointVal[0]*10000);
      robot_targetJointVal[1] = int(targetJointVal[1]*10000);
      robot_targetJointVal[2] = int(targetJointVal[2]*10000);
      robot_targetJointVal[3] = int(targetJointVal[3]*10000);
      robot_targetJointVal[4] = int(targetJointVal[4]*10000);
      robot_targetJointVal[5] = int(targetJointVal[5]*10000);
   }

protected: void buttonSendCartesianClicked()
   {
      string target1 = entryTargetDX.get_text();
      string target2 = entryTargetDY.get_text();
      string target3 = entryTargetDZ.get_text();
      string target4 = entryTargetRX.get_text();
      string target5 = entryTargetRY.get_text();
      string target6 = entryTargetRZ.get_text();
      cartesianTarget[0] = stod(target1);
      cartesianTarget[1] = stod(target2);
      cartesianTarget[2] = stod(target3);
      cartesianTarget[3] = stod(target4);
      cartesianTarget[4] = stod(target5);
      cartesianTarget[5] = stod(target6);
      valid_ik = Robot.InverseKinematics( cartesianTarget, theta_ik );
      robot_theta_ik[0] = int(theta_ik[0]*10000);
      robot_theta_ik[1] = int(theta_ik[1]*10000);
      robot_theta_ik[2] = int(theta_ik[2]*10000);
      robot_theta_ik[3] = int(theta_ik[3]*10000);
      robot_theta_ik[4] = int(theta_ik[4]*10000);
      robot_theta_ik[5] = int(theta_ik[5]*10000);
```

```
        std::string response = (valid_ik)? "Yes" : "No";
        ik_sent = TRUE;
        cout<<"Inverse kinematics ok? "<<response<<endl;
        cout<<"GUI "<<theta_ik[0]<<" "<<theta_ik[1]<<" "<<theta_ik[2]<<"
"<<theta_ik[3]<<" "<<theta_ik[4]<<" "<<theta_ik[5]<<endl;
    }

protected: void buttonTestCartesianClicked()
    {
        valid_ik = Robot.InverseKinematics( FW, theta_ik );
        std::string response = (valid_ik)? "Yes" : "No";
        cout<<"Inverse kinematics ok? "<<response<<endl;
        cout<<"GUI "<<theta_ik[0]<<" "<<theta_ik[1]<<" "<<theta_ik[2]<<"
"<<theta_ik[3]<<" "<<theta_ik[4]<<" "<<theta_ik[5]<<endl;
    }
```

## Appendix F: GUI Send Joint Position Limits Button

```
protected: void buttonLimitsTargetClicked()
   {
      string j1n = entryLimitNJ1.get_text();
      string j2n = entryLimitNJ2.get_text();
      string j3n = entryLimitNJ3.get_text();
      string j4n = entryLimitNJ4.get_text();
      string j5n = entryLimitNJ5.get_text();
      string j6n = entryLimitNJ6.get_text();
      string j1p = entryLimitPJ1.get_text();
      string j2p = entryLimitPJ2.get_text();
      string j3p = entryLimitPJ3.get_text();
      string j4p = entryLimitPJ4.get_text();
      string j5p = entryLimitPJ5.get_text();
      string j6p = entryLimitPJ6.get_text();
      lowPosLim[0] = stod(j1n);
      lowPosLim[1] = stod(j2n);
      lowPosLim[2] = stod(j3n);
      lowPosLim[3] = stod(j4n);
      lowPosLim[4] = stod(j5n);
      lowPosLim[5] = stod(j6n);
      upPosLim[0] = stod(j1p);
      upPosLim[1] = stod(j2p);
      upPosLim[2] = stod(j3p);
      upPosLim[3] = stod(j4p);
      upPosLim[4] = stod(j5p);
      upPosLim[5] = stod(j6p);
      printf("Sending Limits...\n");
   }
```

# Appendix G: GUI Send Cartesian Position Limits Button

```
protected: void buttonLimitsCartesianClicked()
   {
      string dxn = entryLimitNDX.get_text();
      string dyn = entryLimitNDY.get_text();
      string dzn = entryLimitNDZ.get_text();
      string rxn = entryLimitNRX.get_text();
      string ryn = entryLimitNRY.get_text();
      string rzn = entryLimitNRZ.get_text();
      string dxp = entryLimitPDX.get_text();
      string dyp = entryLimitPDY.get_text();
      string dzp = entryLimitPDZ.get_text();
      string rxp = entryLimitPRX.get_text();
      string ryp = entryLimitPRY.get_text();
      string rzp = entryLimitPRZ.get_text();
      lowCPosLim[0] = stod(dxn);
      lowCPosLim[1] = stod(dyn);
      lowCPosLim[2] = stod(dzn);
      lowCPosLim[3] = stod(rxn);
      lowCPosLim[4] = stod(ryn);
      lowCPosLim[5] = stod(rzn);
      upCPosLim[0] = stod(dxp);
      upCPosLim[1] = stod(dyp);
      upCPosLim[2] = stod(dzp);
      upCPosLim[3] = stod(rxp);
      upCPosLim[4] = stod(ryp);
      upCPosLim[5] = stod(rzp);
      printf("Sending Limits...\n");
   }
```

# Appendix H: SOEM RT Runner Aid Functions

```
void add_timespec(struct timespec *ts, int64 addtime)
{
  int64 sec, nsec;
  nsec = addtime % NSEC_PER_SEC;
  sec = (addtime - nsec) / NSEC_PER_SEC;
  ts->tv_sec += sec;
  ts->tv_nsec += nsec;
  if ( ts->tv_nsec >= NSEC_PER_SEC )
  {
    nsec = ts->tv_nsec % NSEC_PER_SEC;
    ts->tv_sec += (ts->tv_nsec - nsec) / NSEC_PER_SEC;
    ts->tv_nsec = nsec;
  }
}

void ec_sync(int64 reftime, int64 cycletime , int64 *offsettime)
{
  static int64 integral = 0;
  int64 delta;
  /* set linux sync point 50us later than DC sync, just as example */
  delta = (reftime - 50000) % cycletime;
  if(delta> (cycletime / 2)) { delta= delta - cycletime; }
  if(delta>0){ integral++; }
  if(delta<0){ integral--; }
  *offsettime = -(delta / 100) - (integral / 20);
  gl_delta = delta;
}
```

# Appendix I: SOEM RT Runner Thread

```
OSAL_THREAD_FUNC_RT runner(void *ptr)
{
   if (ec_init(ifname))
   {
     if ( ec_config_init(FALSE) > 0 )
     {
       int jsRet = 0;
       boolean STOP_runner = FALSE;
       // Motor initialization
       CubemarsMotor cubemars1((uint16) 2,(int) 2);
       CubemarsMotor cubemars2((uint16) 3,(int) 3);
       CubemarsMotor cubemars3((uint16) 4,(int) 4);
       CubemarsMotor cubemars4((uint16) 5,(int) 5);
       CubemarsMotor cubemars5((uint16) 6,(int) 6);
       CubemarsMotor cubemars6((uint16) 7,(int) 7);
       cubemars1.ConfigurePDOs();
       cubemars2.ConfigurePDOs();
       cubemars3.ConfigurePDOs();
       cubemars4.ConfigurePDOs();
       cubemars5.ConfigurePDOs();
       cubemars6.ConfigurePDOs();
       CubemarsMotor::ConfigureMotors();
       dorun = 1;
       int i = 1;
       int j = 1;
       int moveCounter= 0;
       // RT loop time control setup
       struct timespec   ts, tleft;
       int ht;
       int64 cycletime;
       int flip = 0;
       clock_gettime(CLOCK_MONOTONIC, &ts);
       ht = (ts.tv_nsec / 1000000) + 1; /* round to nearest ms */
       ts.tv_nsec = ht * 1000000;
       if (ts.tv_nsec >= NSEC_PER_SEC) {
          ts.tv_sec++;
          ts.tv_nsec -= NSEC_PER_SEC;
       }
       cycletime = *(int*)ptr * 1000; /* cycletime in ns */
       toff = 0;
       while(STOP_runner == FALSE)
       {
          i++;
```

```
/* calculate next cycle start */
add_timespec(&ts, cycletime + toff);
/* wait to cycle start */
clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &ts, &tleft);
wkc = ec_receive_processdata(EC_TIMEOUTRET);
if(wkc >= expectedWKC)
{
   cubemars1.WritingPDOs();
   cubemars2.WritingPDOs();
   cubemars3.WritingPDOs();
   cubemars4.WritingPDOs();
   cubemars5.WritingPDOs();
   cubemars6.WritingPDOs();
   cubemars1.ReadingPDOs();
   cubemars2.ReadingPDOs();
   cubemars3.ReadingPDOs();
   cubemars4.ReadingPDOs();
   cubemars5.ReadingPDOs();
   cubemars6.ReadingPDOs();
   if(STOP_ECAT == TRUE)
   {
      cubemars1.SetTargetVelocity((int32_t) 0);
      cubemars2.SetTargetVelocity((int32_t) 0);
      cubemars3.SetTargetVelocity((int32_t) 0);
      cubemars4.SetTargetVelocity((int32_t) 0);
      cubemars5.SetTargetVelocity((int32_t) 0);
      cubemars6.SetTargetVelocity((int32_t) 0);
      cubemars1.SetControlWord((uint16_t) 0x07);
      cubemars2.SetControlWord((uint16_t) 0x07);
      cubemars3.SetControlWord((uint16_t) 0x07);
      cubemars4.SetControlWord((uint16_t) 0x07);
      cubemars5.SetControlWord((uint16_t) 0x07);
      cubemars6.SetControlWord((uint16_t) 0x07);
      j++;
   }
   if(j >50){
      STOP_runner = TRUE;
   }
   if(i >= 100 && i <=140)
   {
      cubemars1.SetControlWord((uint16_t) 0x07);
      cubemars2.SetControlWord((uint16_t) 0x07);
      cubemars3.SetControlWord((uint16_t) 0x07);
      cubemars4.SetControlWord((uint16_t) 0x07);
      cubemars5.SetControlWord((uint16_t) 0x07);
      cubemars6.SetControlWord((uint16_t) 0x07);
```

```
    cubemars1.SetModeOfOperation((int8_t) 0x08);
    cubemars2.SetModeOfOperation((int8_t) 0x08);
    cubemars3.SetModeOfOperation((int8_t) 0x08);
    cubemars4.SetModeOfOperation((int8_t) 0x08);
    cubemars5.SetModeOfOperation((int8_t) 0x08);
    cubemars6.SetModeOfOperation((int8_t) 0x08);
    cubemars1.SetTargetPosition((int32_t) robot_targetJointVal[0]);
    cubemars2.SetTargetPosition((int32_t) robot_targetJointVal[1]);
    cubemars3.SetTargetPosition((int32_t) robot_targetJointVal[2]);
    cubemars4.SetTargetPosition((int32_t) robot_targetJointVal[3]);
    cubemars5.SetTargetPosition((int32_t) robot_targetJointVal[4]);
    cubemars6.SetTargetPosition((int32_t) robot_targetJointVal[5]);
    cubemars1.SetTargetVelocity((int32_t) 2000);
    cubemars2.SetTargetVelocity((int32_t) 2000);
    cubemars3.SetTargetVelocity((int32_t) 2000);
    cubemars4.SetTargetVelocity((int32_t) 2000);
    cubemars5.SetTargetVelocity((int32_t) 2000);
    cubemars6.SetTargetVelocity((int32_t) 2000);
  }

  if(i >= 150 && i <= 170)
  {
    cubemars1.SetControlWord((uint16_t) 0x1f);
    cubemars2.SetControlWord((uint16_t) 0x1f);
    cubemars3.SetControlWord((uint16_t) 0x1f);
    cubemars4.SetControlWord((uint16_t) 0x1f);
    cubemars5.SetControlWord((uint16_t) 0x1f);
    cubemars6.SetControlWord((uint16_t) 0x1f);
  }

  if(i>200)
  {
    if(modeGUI == 1) // GUI Joint Mode
    {
      if(posLimit(targetJointVal))
      {
        cubemars1.SetTargetPosition((int32_t) robot_targetJointVal[0]);
        cubemars2.SetTargetPosition((int32_t) robot_targetJointVal[1]);
        cubemars3.SetTargetPosition((int32_t) robot_targetJointVal[2]);
        cubemars4.SetTargetPosition((int32_t) robot_targetJointVal[3]);
        cubemars5.SetTargetPosition((int32_t) robot_targetJointVal[4]);
        cubemars6.SetTargetPosition((int32_t) robot_targetJointVal[5]);
      }
    }
    else if(modeGUI == 2) // GUI Cartesian Mode
    {
            if(posLimit(theta_ik))
            {
               if(valid_ik == TRUE && ik_sent == TRUE)
               {
                  cubemars1.SetTargetPosition((int32_t)robot_theta_ik[0]);
                  cubemars2.SetTargetPosition((int32_t)robot_theta_ik[1]);
                  cubemars3.SetTargetPosition((int32_t)robot_theta_ik[2]);
                  cubemars4.SetTargetPosition((int32_t)robot_theta_ik[3]);
                  cubemars5.SetTargetPosition((int32_t)robot_theta_ik[4]);
                  cubemars6.SetTargetPosition((int32_t)robot_theta_ik[5]);
                  ik_sent = FALSE;
               }
               else
               {
                  command = "ERROR: Singularity Triggered";
               }
            }
         }
         else if(modeGUI == 3) // JS Mode
         {
            jsReadPast = jsRead;
            jsRead = 0;
            get_input_int8((int) 1, (uint8_t) 0,&jsRead);
            if(moveCounter >3) // Delay 3 till next command
            {
               moveCounter = 0;
            }
            if (moveCounter == 0)
            {
               jsModeControl(jsReadPast,jsRead);
               jsRet=jsUpdatePos(jsCommand);
            }
            if(posLimit(jsIK))
            {
               if(jsRet==1 && moveCounter == 0)
               {
                  cubemars1.SetTargetPosition((int32_t)robot_theta_ik[0]);
                  cubemars2.SetTargetPosition((int32_t)robot_theta_ik[1]);
                  cubemars3.SetTargetPosition((int32_t)robot_theta_ik[2]);
                  cubemars4.SetTargetPosition((int32_t)robot_theta_ik[3]);
                  cubemars5.SetTargetPosition((int32_t)robot_theta_ik[4]);
                  cubemars6.SetTargetPosition((int32_t)robot_theta_ik[5]);
                  moveCounter = 1;
               }
            }
            if (moveCounter >= 1)
```

```
                    {
                       moveCounter++;
                    }
                    jsRet = 0;
                 }
              }
              theta[0]=cubemars1.GetPositionActualValue()/10000.0;
              theta[1]=cubemars2.GetPositionActualValue()/10000.0;
              theta[2]=cubemars3.GetPositionActualValue()/10000.0;
              theta[3]=cubemars4.GetPositionActualValue()/10000.0;
              theta[4]=cubemars5.GetPositionActualValue()/10000.0;
              theta[5]=cubemars6.GetPositionActualValue()/10000.0;
              thetaD[0]=cubemars1.GetVelocityActualValue()/10000.0;
              thetaD[1]=cubemars2.GetVelocityActualValue()/10000.0;
              thetaD[2]=cubemars3.GetVelocityActualValue()/10000.0;
              thetaD[3]=cubemars4.GetVelocityActualValue()/10000.0;
              thetaD[4]=cubemars5.GetVelocityActualValue()/10000.0;
              thetaD[5]=cubemars6.GetVelocityActualValue()/10000.0;
              torque[0]=cubemars1.GetTorqueActualValue()/10000.0;
              torque[1]=cubemars2.GetTorqueActualValue()/10000.0;
              torque[2]=cubemars3.GetTorqueActualValue()/10000.0;
              torque[3]=cubemars4.GetTorqueActualValue()/10000.0;
              torque[4]=cubemars5.GetTorqueActualValue()/10000.0;
              torque[5]=cubemars6.GetTorqueActualValue()/10000.0;
              velTorLimit(thetaD,torque);
              dataRecorder();
              needlf = TRUE;
           }
           if (ec_slave[0].hasdc)
           {
              /* calulate toff to get linux time and DC synced */
              ec_sync(ec_DCtime, cycletime, &toff);
           }
           ec_send_processdata();
        }
        ec_slave[0].state = EC_STATE_INIT;
        ec_writestate(0); // request INIT state for all slaves
        ec_close(); // closes the network
        STOP_ECAT = TRUE;
     }
   }
   else
   {
     printf("No socket connection on %s\nExecute as root\n",ifname);
   }
}
```

## Appendix J: Joystick Control

```
void jsModeControl(int jsPast, int jsPres)
{
   if(jsPast == 63 && jsPres != 63) // motion tracking
   {
      jsCommand = 0;
      if(jsRead == 62) //Fwd
      {
         jsCommand = 1;
      }
      else if(jsRead == 61) //Bwd
      {
         jsCommand = 2;
      }
      if(jsRead == 59) //Lft
      {
         jsCommand = 3;
      }
      else if(jsRead == 55) //Rght
      {
         jsCommand = 4;
      }
   }
   if (jsPres==63) // change mode
   {
      jsCommand = 0;
      if(jsPast == 47) //spUp
      {
         jsMode ++;
         printf("jsRead: %d\n",jsRead);
         printf("jsMode: %d\n",jsMode);
      }
      else if(jsPast == 31) //spDown
      {
         jsMode --;
         printf("jsRead: %d\n",jsRead);
         printf("jsMode: %d\n",jsMode);
      }
      if(jsMode>2)
      {
         jsMode = 0;
         printf("jsMode: %d\n",jsMode);
      }
      else if (jsMode< 0)
```

```
        {
          jsMode = 2;
          printf("jsMode: %d\n",jsMode);
        }
    }
}
```

## Appendix K: Joystick Send Command

```
int jsUpdatePos(int command)
{
   if (command !=0)
   {
      Robot.ForwardKinematics( theta, jsFW );
      printf("Continuous Movement Active %d\n",command);
      if(command == 1) //fwd tilt
      {
         if(jsMode == 0)
         {
            printf("moving forward\n");
            jsFW[0] = jsFW[0] + linStepSize;
         }
         else if(jsMode == 1)
         {
            jsFW[2] = jsFW[2] - linStepSize;
         }
         else if(jsMode == 2)
         {
            jsFW[3] = jsFW[3] + rotStepSize;
         }
      }
      else if(command == 2) //bwd tilt
      {
         if(jsMode == 0)
         {
            jsFW[0] = jsFW[0] - linStepSize;
         }
         else if(jsMode == 1)
         {
            printf("moving up\n");
            jsFW[2] = jsFW[2] + linStepSize;
         }
         else if(jsMode == 2)
         {
            jsFW[3] = jsFW[3] - rotStepSize;
         }
      }
      else if(command == 3) //lft tilt
      {
         if(jsMode == 0)
         {
            jsFW[1] = jsFW[1] + linStepSize;
            }
            else if(jsMode == 1)
            {
               jsFW[5] = jsFW[5] + rotStepSize;
            }
            else if(jsMode == 2)
            {
               jsFW[4] = jsFW[4] + rotStepSize;
            }
         }
         else if(command == 4) //rght tilt
         {
            if(jsMode == 0)
            {
               jsFW[1] = jsFW[1] - linStepSize;
            }
            else if(jsMode == 1)
            {
               jsFW[5] = jsFW[5] - rotStepSize;

            }
            else if(jsMode == 2)
            {
               jsFW[4] = jsFW[4] - rotStepSize;
            }
         }
         valid_ik = Robot.InverseKinematics( jsFW, jsIK );
         if(valid_ik)
         {
            robot_theta_ik[0] = int(jsIK[0]*10000);
            robot_theta_ik[1] = int(jsIK[1]*10000);
            robot_theta_ik[2] = int(jsIK[2]*10000);
            robot_theta_ik[3] = int(jsIK[3]*10000);
            robot_theta_ik[4] = int(jsIK[4]*10000);
            robot_theta_ik[5] = int(jsIK[5]*10000);
            return 1;
         }
         return 0;

      }
   }
```

# Appendix L: EtherCAT Check

```
OSAL_THREAD_FUNC ecatcheck(void *ptr)
{
   int slave;
   printf("Thread on\n");
   while(STOP_ECAT == FALSE)
   {
      if( inOP && ((wkc < expectedWKC) || ec_group[currentgroup].docheckstate))
      {
         if (needlf)
         {
           needlf = FALSE;
           printf("\n");
         }
         /* one ore more slaves are not responding */
         ec_group[currentgroup].docheckstate = FALSE;
         ec_readstate();
         for (slave = 1; slave <= ec_slavecount; slave++)
         {
           if ((ec_slave[slave].group == currentgroup) && (ec_slave[slave].state !=
EC_STATE_OPERATIONAL))
           {
              ec_group[currentgroup].docheckstate = TRUE;
              if (ec_slave[slave].state == (EC_STATE_SAFE_OP + EC_STATE_ERROR))
              {
                printf("ERROR : slave %d is in SAFE_OP + ERROR, attempting ack.\n", slave);
                ec_slave[slave].state = (EC_STATE_SAFE_OP + EC_STATE_ACK);
                ec_writestate(slave);
              }
              else if(ec_slave[slave].state == EC_STATE_SAFE_OP)
              {
                printf("WARNING : slave %d is in SAFE_OP, change to OPERATIONAL.\n",
slave);
                ec_slave[slave].state = EC_STATE_OPERATIONAL;
                ec_writestate(slave);
              }
              else if(ec_slave[slave].state > EC_STATE_NONE)
              {
                if (ec_reconfig_slave(slave, EC_TIMEOUTMON))
                {
                   ec_slave[slave].islost = FALSE;
                   printf("MESSAGE : slave %d reconfigured\n",slave);
                }
              }
```

```
                else if(!ec_slave[slave].islost)
                {
                  /* re-check state */
                  ec_statecheck(slave, EC_STATE_OPERATIONAL, EC_TIMEOUTRET);
                  if (ec_slave[slave].state == EC_STATE_NONE)
                  {
                    ec_slave[slave].islost = TRUE;
                    printf("ERROR : slave %d lost\n",slave);
                  }
                }
              }
              if (ec_slave[slave].islost)
              {
                if(ec_slave[slave].state == EC_STATE_NONE)
                {
                  if (ec_recover_slave(slave, EC_TIMEOUTMON))
                  {
                    ec_slave[slave].islost = FALSE;
                    printf("MESSAGE : slave %d recovered\n",slave);
                  }
                }
                else
                {
                  ec_slave[slave].islost = FALSE;
                  printf("MESSAGE : slave %d found\n",slave);
                }
              }
            }
            if(!ec_group[currentgroup].docheckstate)
              printf("OK : all slaves resumed OPERATIONAL.\n");
          }
          osal_usleep(10000);
      }
  }
```

# Appendix M: Data Logger

```
void dataRecorder()
{
   if(recordData)
   {
      counter4Counter++;
   }
   else
   {
      logCounter = 0;
      counter4Counter = 0;
   }
   if(counter4Counter>=sampleFactor)
   {
      logCounter++;
      counter4Counter = 0 ;
      if(logCounter == 1)
      {
         printf("started logging %d \n",logCounter);
         csvfile csv(logFileName,false); // throws exceptions!
         // Hearer
         csv <<"Time" << "J1_pos" << "J2_pos" << "J3_pos" << "J4_pos" << "J5_pos" <<
"J6_pos" << "J1_vel" << "J2_vel" << "J3_vel" << "J4_vel" << "J5_vel" << "J6_vel" <<
"J1_tor" << "J2_tor" << "J3_tor" << "J4_tor" << "J5_tor" << "J6_tor" << "J1_tpos" <<
"J2_tpos" << "J3_tpos" << "J4_tpos" << "J5_tpos" << "J6_tpos" << endrow;
         logCounter++;
      }
      else if(logCounter>1)
      {
         csvfile csv(logFileName,true); // throws exceptions!
         // Hearer
         if(modeGUI == 1)
         {
            csv <<logCounter*timeFactor <<  theta[0]  << theta[1]   << theta[2]     << theta[3]
<< theta[4]   << theta[5]  <<  thetaD[0] << thetaD[1]  << thetaD[2]    << thetaD[3] <<
thetaD[4]  << thetaD[5] << torque[0] << torque[1]  << torque[2]    << torque[3] << torque[4]
<< torque[5] <<  targetJointVal[0] << targetJointVal[1] << targetJointVal[2] <<
targetJointVal[3] << targetJointVal[4] << targetJointVal[5] <<  endrow;
         }
         else if(modeGUI == 2)
         {
            csv <<logCounter*timeFactor <<  theta[0]    << theta[1]     << theta[2]     <<
theta[3]     << theta[4]      << theta[5]      <<  thetaD[0]    << thetaD[1]     << thetaD[2]     <<
thetaD[3]     << thetaD[4]     << thetaD[5]     << torque[0]    << torque[1]     << torque[2]     <<
torque[3]    << torque[4]    << torque[5]    <<  theta_ik[0] << theta_ik[1]  << theta_ik[2]  <<
theta_ik[3]  << theta_ik[4]  << theta_ik[5]  <<  endrow;
            }
            else if(modeGUI == 3)
            {
                csv <<logCounter*timeFactor <<  theta[0]  << theta[1]   << theta[2]     << theta[3]
<< theta[4]   << theta[5]  <<  thetaD[0] << thetaD[1]  << thetaD[2]    << thetaD[3] <<
thetaD[4]  << thetaD[5] << torque[0] << torque[1]  << torque[2]    << torque[3] << torque[4]
<< torque[5] <<  jsIK[0] << jsIK[1]  << jsIK[2]  << jsIK[3]  << jsIK[4]  << jsIK[5]  <<
endrow;
            }
        }
    }
}
```

# Appendix N: Angular and Cartesian Limit Protection

```
int posLimit(double posR[6])
{
   double cPos[6];
   if(posR[0]<lowPosLim[0] || posR[0]>upPosLim[0] ||
     posR[1]<lowPosLim[1] || posR[1]>upPosLim[1] ||
     posR[2]<lowPosLim[2] || posR[2]>upPosLim[2] ||
     posR[3]<lowPosLim[3] || posR[3]>upPosLim[3] ||
     posR[4]<lowPosLim[4] || posR[4]>upPosLim[4] ||
     posR[5]<lowPosLim[5] || posR[5]>upPosLim[5])
   {
      return 0;
   }
   Robot.ForwardKinematics( posR, cPos );
   else if(cPos[0]<lowCPosLim[0] || cPos[0]>upCPosLim[0] ||
         cPos[1]<lowCPosLim[1] || cPos[1]>upCPosLim[1] ||
         cPos[2]<lowCPosLim[2] || cPos[2]>upCPosLim[2] ||
         cPos[3]<lowCPosLim[3] || cPos[3]>upCPosLim[3] ||
         cPos[4]<lowCPosLim[4] || cPos[4]>upCPosLim[4] ||
         cPos[5]<lowCPosLim[5] || cPos[5]>upCPosLim[5])
   {
      return 0;
   }
   else
   {
      return 1;
   }
}
```

## Appendix O: Velocity and Torque Limit Protection

```
void velTorLimit(double velR[6],double torR[6])
{
   if(velR[0]<lowVelLim[0] || velR[0]>upVelLim[0] ||
      velR[1]<lowVelLim[1] || velR[1]>upVelLim[1] ||
      velR[2]<lowVelLim[2] || velR[2]>upVelLim[2] ||
      velR[3]<lowVelLim[3] || velR[3]>upVelLim[3] ||
      velR[4]<lowVelLim[4] || velR[4]>upVelLim[4] ||
      velR[5]<lowVelLim[5] || velR[5]>upVelLim[5])
   {
      limCounter++;
            if(limCounter >=10)
            {
         STOP_ECAT = TRUE;
      }
   }
   else if(torR[0]<lowTorLim[0] || torR[0]>upTorLim[0] ||
         torR[1]<lowTorLim[1] || torR[1]>upTorLim[1] ||
         torR[2]<lowTorLim[2] || torR[2]>upTorLim[2] ||
         torR[3]<lowTorLim[3] || torR[3]>upTorLim[3] ||
         torR[4]<lowTorLim[4] || torR[4]>upTorLim[4] ||
         torR[5]<lowTorLim[5] || torR[5]>upTorLim[5])
   {
      limCounter++;
            if(limCounter >=10)
            {
         STOP_ECAT = TRUE;
      }
   }
   else
   {
      limCounter = 0;
   }
}
```

# Appendix P: Cubemars Motors ORH Parameters

## PARAMTERS

| Basic Data | | | | |
|---|---|---|---|---|
| Model | ORH 40(RR40) | ORH50(RR50) | ORH60(RR60) | ORH70(RR70) |
| OD (mm) | 56mm | 68.4mm | 81mm | 86mm |
| Length | 62.4mm | 63.1mm | 61.2mm | 73mm |
| Hollow Shaft dia (mm) | 9mm | 11mm | 12.5mm | 14mm |
| Weight (g) | 335g | 500g | 650g | 720g |
| Hollow Encoder | 14bit | 14 bit | 14bit | 14bit |
| Expandable 17 bit Encoder for Output shaft | | | | |
| Motor Parameters | | | | |
| Continuous Torque (Nm) | 0.32 | 0.5 | 0.75 | 0.9 |
| Peak Torque (Nm) | 0.95 | 1.5 | 2.3 | 2.6 |
| KV (RPM/V) | 180 | 180 | 115 | 100 |
| Kt (Nm/A) | 0.025408 | 0.0397 | 0.083 | 0.09 |
| Ke (V/RPM) | 0.005555556 | 0.005555556 | 0.008695652 | 0.01 |
| Rated Current (Amp) | 8 | 12.5 | 9 | 10 |
| Peak Current (Amp) | 23 | 37 | 28 | 30 |
| Input Voltage (V) | 48 | 48 | 48 | 48 |
| Number of Pole-Pair | 12 | 14 | 14 | 21 |
| Gear Ratio (GR) | 100 | 100 | 100 | 100 |
| Rated Torque @2000 RPM (Nm) with GR 100 | 5.4 | 16 | 28 | 40 |
| Maximum allowable torque (Nm) at start and stop | 19 | 37 | 57 | 82 |
| Instantaneous allowable maximum torque | 35 | 71 | 95 | 147 |
| Maximum allowable average load torque (Nm) | 7.7 | 27 | 34 | 49 |
| Maximum allowable Input Speed | 8000 rpm | 7000 rpm | 6000 rpm | 6000 rpm |
| Allowable Average Input Speed | 3500 rpm | 3500 rpm | 3500 rpm | 3500 rpm |
| Precission | 10 ≤ Arc Sec | 10≤ Arc Sec | 10 ≤ Arc Sec | 10 ≤ Arc Sec |
| Projected Life Span | 15000 hr | 15000 hr | 15000 hr | 15000hr |
| Motor Driver | | | | |
| Input Voltage | 48V | 48V | 48V | 48V |
| EtherCat | yes | yes | yes | yes |
| Hollow Shaft dia (mm) | 11 | 11 | 12.5 | 14 |
| Synchronus position control, velocity control, and torque control | yes | yes | yes | yes |
| EtherCat Ports In and out | yes | yes | yes | yes |
| Power Ports (90 deg radail, horiontal) In and Out | no | yes | yes | yes |