DTU Electro
Department of Electrical and Photonics Engineering

Exploration and Exploitation of the Active Inference Framework for E-Grocery Tasks with a Robotic Arm

Master Thesis

Exploration and Exploitation of the Active Inference Framework for E-Grocery Tasks with a Robotic Arm

Master Thesis
June, 2024

Written by:
Gabriele Rutigliano

Advisor:
Silvia Tolu, Associate Professor, Ph.D.

Cover photo: Vibeke Hempler, 2012
Published by: DTU, Department of Electrical and Photonics Engineering,
Elektrovej, Building 326,
2800 Kgs. Lyngby Denmark
https://electro.dtu.dk/

ECTS: 30

Approval

This thesis has been prepared over five months at the Department of Electrical and Photonics Engineering, at the Technical University of Denmark, DTU, in partial fulfillment for the degree Master of Science in Automation and Robotics Engineering, M.Sc.

It is assumed that the reader has a basic knowledge in the areas of control theory and robotics.

Gabriele Rutigliano - s212519

Signature

02/06/2024

Date

Acknowledgements

I would like to express my gratitude to the following people for their immense help and support throughout the development of my thesis.

Silvia Tolu
Associate Professor, Department of Electrical and Photonics Engineering Automation and Control, Supervisor
Your inputs and suggestions have provided great guidance, helping me to approach topics and challenges I had never faced before.

Lisa
Best friend, fiancée
Who, at this point, probably knows this thesis better than I do. Thank you for bearing with me these very long five months (and more), and for constantly believing in me, even when everything in this project seemed to be going wrong.

Alessia R.
Best friend
Thank you for supporting me and keeping me sane, even from afar and for matters that did not strictly involved this thesis but would have otherwise blocked me.

My Family
For staying close to me, believing in me, and supporting me throughout my entire academic journey. I will always appreciate this.

Abstract

This study aims to compare the performance of a model-free adaptive controller, the unbiased Active Inference Controller (uAIC), against the Active Inference Controller (AIC) and the Model Reference Adaptive Controller (MRAC) in the context of e-grocery applications.

Performance was evaluated through tasks designed to test tracking in both joint-space and task-space, as well as in a simulated e-grocery environment with static and moving objects. An inverse kinematic solver, Trac-IK, was used to convert task-space references to joint-space for the controllers.

During the implementation phase of the uAIC, it was found that, to work properly with the added weight of the gripper, the controller needed a re-tuning of its control parameters. This was unexpected for a model-free adaptive controller.

With the uAIC controller re-tuned to achieve satisfactory response, the tests showed that the AIC outperformed the MRAC and the uAIC in static and dynamic reference tracking in task-space. However, the uAIC demonstrated lower tracking errors at higher speeds for time-varying references and showed superior performance in grasping moving objects in an e-grocery setup.

The study suggests that the uAIC has significant potential advantages over the AIC for e-grocery applications, particularly in dynamic environments, although further research is needed to optimize its performance.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context and Motivation

To facilitate the widespread adoption of automation technology, especially in rapidly evolving industries, it is essential to develop robotic systems that offer easy reconfiguration and adaptability to shifting manufacturing requirements and product variations. Overcoming one of the primary challenges associated with Robot Process Automation (RPA), which typically demands standardized processes and consistent input data, is critical. RPA systems excel in well-defined tasks with constant task conditions and well-controlled or eliminated external disturbances [1]. Manual labor, however, remains preferable when task conditions are variable.

Nevertheless, there is a growing industry need for robot controllers that exhibit greater flexibility and adaptability to real-time variations. Robot manipulators frequently operate in environments with dynamic changes, encountering noisy sensory inputs and unexpected events, especially in tasks like pick-and-place, where the dynamics can shift unpredictably dealing with unknown objects or those with significant variations in weight and shape.
An example of this can be seen in the food industry. Fruits and vegetables come in various shapes and sizes, which complicates the choice of adequate manipulators and controllers that can handle different types effectively. Manipulators must be adaptable to these variations without requiring extensive reconfiguration [2].

Closing the efficiency divide between RPA systems and human-like adaptability is crucial for advancing technology adoption. Drawing inspiration from biological systems and using existing cognitive models can give industrial robots human-like abilities. A tangible illustration of this concept is found in the Active Inference Control (AIC). The assumption behind active inference in motor control is that, given a specific internal model of the environment, a set of sensory inputs can characterize how that model could be generated [3]. Therefore, the causes of sensory data can be inferred.

Given that the environment influences sensory information in humans, the act of taking action implies that humans are influencing the environment, consequently affecting sensory input. From this perspective, motor control can be described as the realization of pre-existing expectations concerning proprioceptive sensations [3]. In practical terms, when reaching for an object without knowing its properties, such as weight, a guess is made about how much force is needed based on the internal model of the world. If this expectation is not met during the attempt to move the object, adjustments are made instinctively and responsively. This adaptive nature of control is highly desirable in robotic applications when dealing with unknown object parameters and robot dynamics.

Currently AIC has already been applied in various context and different robotic application, that is also the case for the Franka Emika Panda co-bot [3]. However, a novel version of AIC, the Unbiased Active Inference Control (uAIC), brings new improvements that can benefit robotic operations in the food industry [4]. The controllers will be deployed on a 7 Degrees of Freedom (DoF) robotic arm, the Franka Emika Panda robot, and their performances will be assessed by running a series of tasks and comparing results.

1.2 Prior Work

A description and analysis of both the AIC and uAIC can be found in [4], and a more extensive comparison with other type of controllers can also be found in [5].
Moreover, this project will build upon the work done by the master student Christian Kampp Kruuse in his master thesis [6] by leveraging his implementation of the Inverse Kinematics (IK) resolver: TRAC-IK, onto the Franka Emika Panda robot to be able to work in task-space.
Furthermore, I will incorporate findings from a literature study undertaken in a previous project course [7], as well as an implementation of an e-grocery task conducted within a task-space designed and developed in said project course [7]. These elements will be presented and reused in the context of the thesis report.

1.3 Research Questions

The controller, developed by [4] represents an update over the AIC previously deployed by the same authors in [3].
More specifically it was seen in [4] that the uAIC performed better in terms of position and velocity tracking in joint-space, and also in terms of the behaviour in case of collision, providing a much smoother and less aggressive control action when compared to the AIC. These improvements were deemed beneficial if the robot were to handle delicate objects, such as fruits and vegetables.
Thus, some questions can be raised:

- Will the use of an improved version of an AIC bring any measurable improvement to real-world applications when compared to its predecessor?
- How do these improvements benefit a real-world static and dynamic task in the e-grocery?

To respond to these two questions two sets of task will be put in place.

First, a static and dynamic test task will analyse the performance of the two controllers. The performance metric that will be used in this context will be the position tracking error along the x, y and z axes, the joint-space tracking error and the spent torque.

Second, both a static object grasping task and a moving object grasping task will be implemented to analyze the performance of the two controllers in an e-grocery context. The performance metrics used in this context will be the success rate in completing these tasks and the time required to do so.

1.4 Research Goals and Methods

The goal of this project is to deploy an adaptive architecture for a robotic manipulator system capable of effectively handling static and dynamic industrial pick-and-place tasks. To achieve this the uAIC will be employed.
Following the literature review, the aim is to propose a novel system architecture for an e-grocery pick and place task.
Transitioning to the implementation phase, the project involves deploying the controller onto the robot and developing various tasks in Robot Operating System (ROS). Subsequently, the performances of both the uAIC and AIC will be assessed using the performance metrics defined here, after gathering the necessary data.
Lastly, the findings will be documented, and insights on further development in the field

will be provided.
In summary:

- Perform a literature study of topics relevant to the thesis work;
- Propose and implement a novel adaptive robot manipulator control solution exploiting a bio inspired control approach (uAIC);
- Test and validate the implemented solution in a real world application and compare findings with the main hypothesis.
- Formulate conclusions regarding the effective improvements over using the previous AIC in the context of an e-grocery application.

1.5 Thesis outline

First, the literature review introduces the current needs and challenges in the food industry, and more specifically in the e-grocery, as well as the control approaches that will be investigated in this project.
Following this, Chapter 3 provides an overview of the hardware and software used throughout the project.
Chapter 4 outlines the design and implementation process, detailing the approach taken to build the final system for this project.
Next, Chapter 5 presents the results obtained from the conducted experiments.
In Chapter 6, the performance of the controllers is analyzed, including the effects of their interaction with other components of the system and the respective influence on overall performance.
Finally, the research question is answered, and the main conclusions from the discussion are summarized.

2 Literature Review

This chapter provides an introduction to the needs and challenges in the food industry, especially in e-grocery, as well as an overview of the controllers relevant to this thesis project, with a focus on AIC and the newer version, uAIC.
Emphasis will be placed on analyzing the structural differences between these controllers. Additionally, a brief overview of a more classical control approach, the Model Reference Adaptive Controller (MRAC), will be included.

2.1 Automation Challenges in the Food Industry and E-Grocery

This chapter reviews the current needs and challenges in the food industry, with a specific focus on e-grocery. The food industry includes all activities involved in producing, processing, distributing, and selling food products. This ranges from farming and manufacturing to retail and food services.
E-grocery is a part of the food industry that deals with the online sale and home delivery of groceries.

The e-grocery sector faces significant challenges in automating processes. These challenges arise from the inherent variability and fragility of food products, the unstructured environments in which they are managed, and the necessity for high-speed yet precise operations. For instance, fruits and vegetables exhibit significant differences in shape, size, and weight, necessitating advanced robotic manipulators capable of dynamic adaptation. Ensuring that these delicate items are not damaged during handling adds an additional layer of complexity.

Robotic manipulators must navigate the delicate balance between applying enough force to handle produce securely while avoiding damage. For instance, research has shown that handling fragile fruits and vegetables requires advanced sensory feedback and control systems to dynamically adjust the grip force [8]. Additionally, the variability in shape and size necessitates adaptable systems that can manage different types of produce without extensive reconfiguration [2].

Striking the right balance between efficiency and care remains a significant challenge. High-speed operations must be balanced with the need for precision to prevent damage to delicate produce. This is particularly challenging in e-grocery, where consumer expectations for quality and speed are high. As the demand for automated solutions grows, overcoming these challenges is essential for enhancing operational efficiency and maintaining the quality of food products in the e-grocery industry. Continued research and development in robotic manipulation, sensor integration, and control systems are critical to address these issues and advancing the automation of food handling processes [9].

2.2 Overview of Control Approaches

In this section, we provide an overview of different control approaches that are relevant to this thesis project. As mentioned in Chapter 1, this report investigates the performances of the newer uAIC when placed in a context of an e-grocery. These performances will be compared to those obtained by the AIC, which will be used as the benchmark. Additionally,

since also the MRAC was made available, thanks do the work conducted by Christian Kampp Kruuse in [6], this controller will also be used to perform the tasks and record results but focus will be given to the comparison between the AIC and uAIC.

2.2.1 The Active Inference Controller

2.2.1.1 Foundation of Active Inference

Active Inference (AIF) is a biologically plausible mathematical construct based on the Free Energy Principle (FEP) proposed by Karl Friston [10]. This principle describes how living systems resist a natural tendency to disorder. It posits that biological systems, including the human brain, operate by minimizing free energy, which is a measure of surprise or prediction error. In other words, organisms strive to reduce the difference between their predictions about the world and the actual sensory input they receive [10].

Current implementation of AIF in robotic applications has been brought out in state-estimation, control, planning and high-level cognitive skills [11]. In the context of this project however emphasis is placed in the use of active inference for state-estimation and control. To this regard of high importance is the work conducted in [3] where the authors presented an adaptive scheme for controlling a generic n-DoFs robot manipulators, namely the (AIC).

2.2.1.2 Structure and Function of the AIC

As mentioned, this project will implement the AIC controllers presented in [3] and [4]. These controllers are model-free, meaning that they are not provided with a model of the system and the environment to base their state estimate on. Instead, they create and maintain a model of the system and the environment, continuously updating their beliefs based on incoming sensory data to minimize prediction errors.

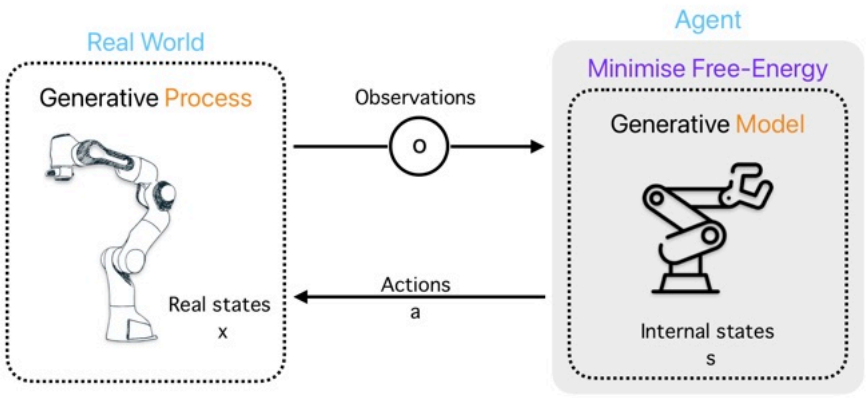


Figure 2.1: High-level visualization of the working principle of an AIC controller

Figure 2.1 shows the working principle of an AIC controller at a very high level.
As it can be noticed the controller operates through a cyclic process that involves the following steps:

1. **Initialization:** The AIC starts with an initial probabilistic model of the environment.
2. **Prediction:** At each time step, the AIC generates predictions of expected sensory inputs based on its current state.
3. **Observation:** The AIC receives actual sensory inputs from the environment.

4. **Comparison:** The predicted inputs are compared with actual inputs to calculate prediction errors.
5. **Belief Update:** The AIC updates its beliefs to minimize prediction errors, refining its model of the environment.
6. **Action Selection:** Actions are chosen to further minimize prediction errors in future observations.
7. **Execution:** The selected actions are executed, influencing the environment and leading to new sensory inputs.

This iterative process allows the AIC to continuously adapt and improve its model, ensuring effective control even in dynamic and uncertain environments.

The AIC presented in [3] is a torque controller that uses proprioceptive sensors for position and velocity to control joint positions. The control law proposed is generalized for any n-DoF manipulator and has been specifically implemented on a 7-DoF robot arm, the Franka Emika Panda, in joint-space in[3].

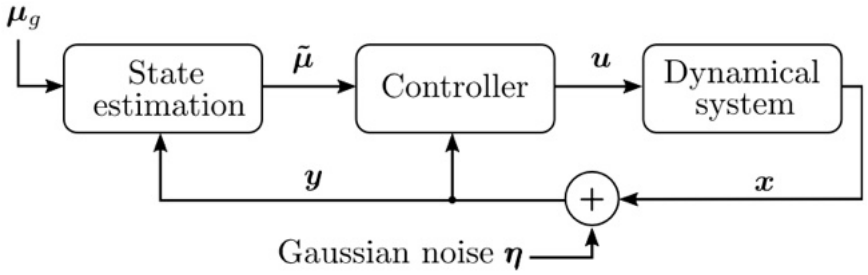


Figure 2.2: Control diagram of the AIC [4]

In Figure 2.2 an overview of the control diagram implemented in [3] can be seen. The symbology used is summarised in table Table 2.1.

Symbol	Description
μ_g	The desired goal. A $\mathbb{R}^7$ vector that contains the desired joint positions.
$\tilde{\mu}$	Generalised motions vector, up to the third order. A set of $\mathbb{R}^7$ vectors containing the beliefs about joint positions, velocities and accelerations.
u	Control actions. A $\mathbb{R}^7$ vector that contains the commanded torques.
x	States of the system. A $\mathbb{R}^7$ vector containing the joint positions of the robotic arm.
y	Retrieved sensory input. A set of two $\mathbb{R}^7$ vectors which contains the measured joint position and velocities.

Table 2.1: Table of the symbology used to describe the AIC control diagram

The desired goal, μ_g, is used for state estimation. This vector contains the desired joint positions for each of the seven joints of the robotic manipulator. Encoding the goal in the prior (the assumed knowledge about the state of the system) will cause a biased state estimation, as will be described later in this section when introducing the uAIC.

The state estimator evaluates the estimates of the system states and their derivatives. To describe the dynamics of the beliefs about the system states, the authors of [3] utilize the concept of generalized motion from [12]. Specifically, "generalized motions" refer to a comprehensive representation of the system's dynamics, including the states and their n-th order derivatives. In [3], the authors use up to the second-order derivatives to describe the beliefs about the system's dynamics, as the robotic manipulator only provided position and velocity sensors. Thus, these were the only predicted dynamics that could be compared to the actual measured dynamics of the robot.

The estimates obtained are then used by the controller to evaluate the control actions u that steer the system to the desired state while minimizing the prediction errors in F. This is achieved using gradient descent.

The control actions affect the system states and, consequently, the measurements obtained by the sensors. The authors in [3] assume that the noise affecting the sensor readings is Gaussian and uncorrelated. This is modeled as Gaussian noise η entering the feedback loop and affecting y.

In the AIC, the control law is calculated as a sum of sensory prediction errors, similar to how Proportional-Integral (PI) controllers work, where the control output is directly influenced by the current error.
Specifically the proportional (P) gain is driven by errors in the velocity sensory prediction, whereas the integral (I) gain is affected by the position sensory prediction

2.2.2 The Unbiased Active Inference Controller

The controller setup just presented has several drawbacks as elaborated by the authors in [4]. Hence, a newer version was developed: the uAIC [4].
The main drawbacks of the AIC are:

1. **Biased State Estimation:** The AIC focuses on the relationship between states and observations without directly modeling control actions. In traditional filters like the Kalman filter, the prior is derived from a prediction step that uses the previous state and control action. However, in the AIC, the prior inherently assumes the agent will move towards the target, leading to a biased state estimate [4]. On the other hand, the uAIC has an unbiased belief over the state [4].
2. **Implicit Modelling of Actions:** The actions of the robot are not directly specified or modeled straightforwardly. Instead, the actions (torques) in the AIC are computed using gradient descent, a method to adjust the robot's actions to minimize errors. However, since the robot's actions are not directly modeled, the chain rule is employed to link changes in its actions to changes in its outcomes.

   While this simplification makes calculations easier, it can limit the robot's performance, especially in environments with complex, non-linear dynamics [4]. The control law of the AIC is driven by sensory prediction errors, which are differences between what the robot's sensors predict and what is actually measured by the robot's

sensors. These errors are weighted by their precision, indicating how much trust is placed in each sensor's prediction. Essentially, the control law functions similarly to a PI controller.

However, integral control has some drawbacks. If the robot encounters an obstacle and cannot reach its goal, the control effort keeps increasing without achieving the goal, leading to integrator windup. Additionally, because integral control relies on sensory information, a fault in the sensors can cause the control actions to become unpredictable. Consequently, if part of the system is unobserved, it cannot be effectively controlled [4].

The uAIC on the other hand has a more traditional flow of information. State estimation requires the measurements y. Then the (unbiased) belief μ_x and the goal μ_g are fed into the controller which produces the control action as seen in Figure 2.3.

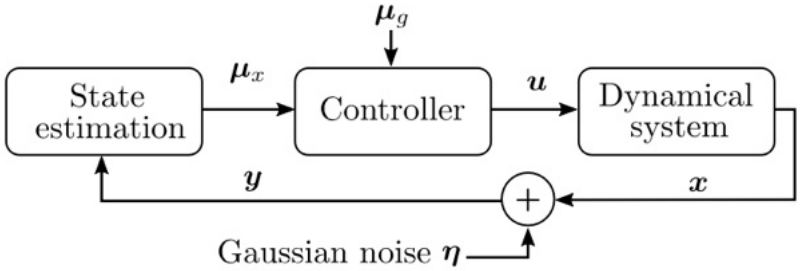


Figure 2.3: Control diagram of the uAIC [4]

Symbol	Description
μ_g	The desired goal. A set of $\mathbb{R}^7$ vectors that contains the desired joint positions and velocities.
$\tilde{\mu}$	Generalised motions vector, up to the second order. A set of $\mathbb{R}^7$ vectors containing the beliefs about joint positions and velocities.
u	Control actions. A $\mathbb{R}^7$ vector that contains the commanded torques.
x	States of the system. A set of $\mathbb{R}^7$ vector containing the joint positions and velocities of the robotic arm.
y	Retrieved sensory input. A set of two $\mathbb{R}^7$ vectors which contains the measured joint position and velocities.

Table 2.2: Table of the symbology used to describe the uAIC control diagram

In contrast to the AIC, in the uAIC, the control actions are explicitly modeled, and the target state is encoded in the control term. The uAIC has a control law irrespective of the number of sensors [4].

In the uAIC, state estimation and control actions are achieved by gradient descent along the free-energy. With this new formulation of the probabilistic model and free-energy, the authors of [4] have been able to prove the convergence of the estimate towards the real state, without any bias towards the goal. Further details about

2.2.2.1 Comparison of AIC and uAIC in a Simulated Environment

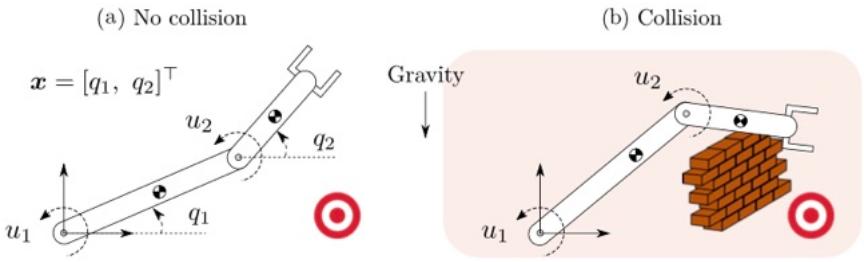


Figure 2.4: Scenario used to illustrate the incorrect state estimation due to the bias toward the target position [4]

In [4] the researchers have simulated the scenario depicted in Figure 2.4. In this scenario the robot has to reach a target, but the path is obstructed by an obstacle. The collision persists for $t_c = 3s$ and this allows to show the incorrect state estimation that takes place with the AIC. The simulated 2-DOF robot arm is equipped with position and velocity sensors, and both are assumed to be affected by zero mean Gaussian noise. The simulation results can be observed in Figure 2.5.

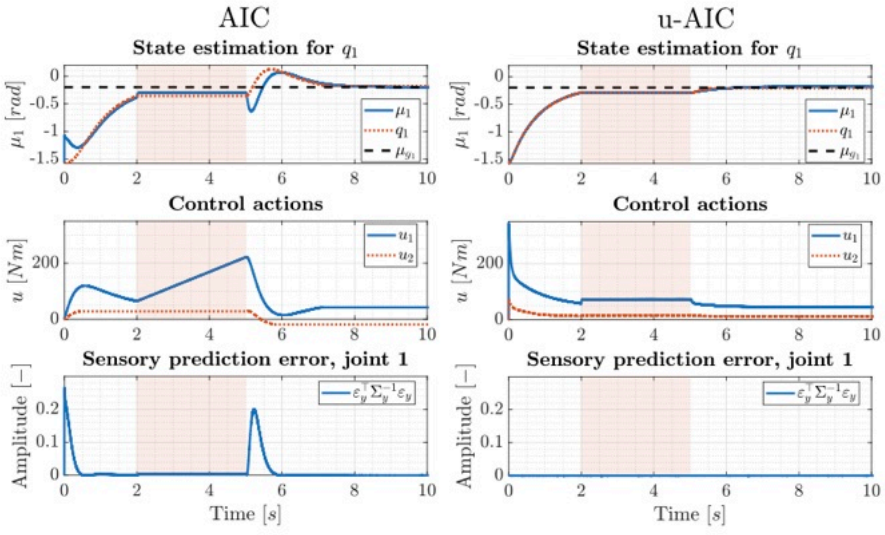


Figure 2.5: Simulation results of a 2-DOF robot arm during a collision [4]

From Figure 2.5 it is possible to notice two things [4]:

1. **Biased state estimation:** In the first row of the plots it is possible to notice how with the AIC, during the collision (red area of the plot) the belief μ_1 does not follow

the real state q_1, whereas the uAIC converges to the true state;

2. **Monotonic increase of control input:** The second row of the plot also shows the control input for both the AIC and uAIC. For the AIC it is possible to see that during collision, due to the integral nature of the AIC, the control input increases. Once the obstacle is removed the controller necessarily overshoots.
   In the uAIC one can only saturate the integral term and not the entire control law. On average, the AIC has four times the overshoot after a collision compared to the uAIC.

2.2.2.2 Comparison of AIC and uAIC on the Franka Emika Panda

On the real Panda arm, the researchers in [4] compared both the reference tracking and the collision behaviour in terms of overshoot and control actions obtained by holding the robotic arm away from its set point.

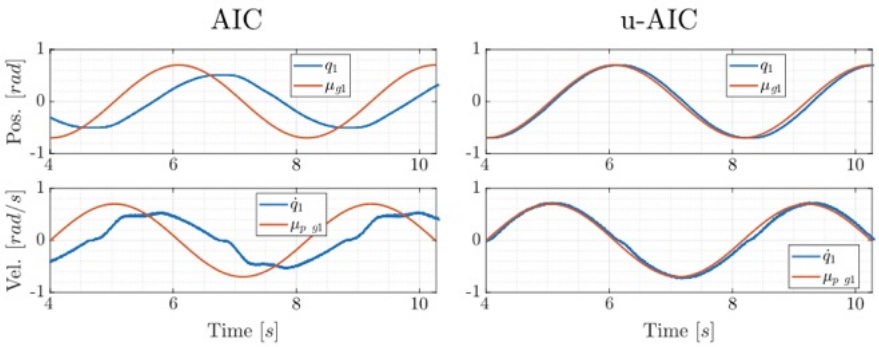


Figure 2.6: Reference tracking on the real Panda arm [4]

As it is seen in Figure 2.6 the reference tracking of a sinusoidal wave has been conducted. The AIC never converges to the reference trajectory, whereas the uAIC does.

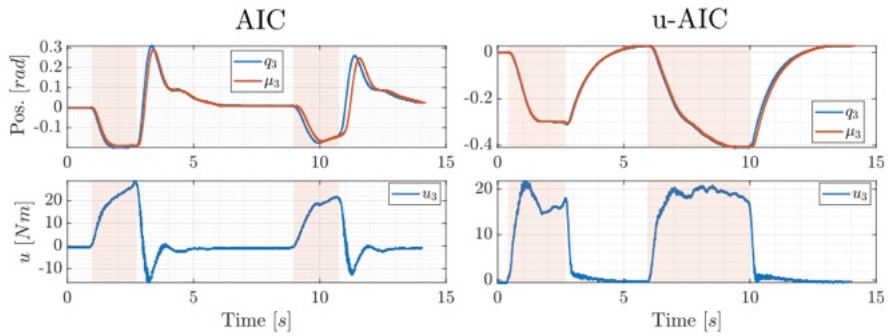


Figure 2.7: Experiment with Panda arm during unwanted interaction (red area) [4]

In Figure 2.7 the collision behaviour is reported. During this experiment the robot had to keep the same position while a person was pushing, pulling and holding the robot.
The AIC showed high overshoots which might be dangerous for people or objects, especially delicate objects. The uAIC, on the other hand, is well-behaved during interaction.

A more in depth comparison was also conducted in [5]. The authors have compared

the Multi-Modal Active Inference Controller (MAIC) with state-of-the-art active inference controllers, such as the AIC [3] and the uAIC [4], and standard controllers, such as Model Predictive Control (MPC) and joint Impedance Control (IC).
In a specific set of experiments the authors in [5] have investigated the adaptability to unmodeled dynamics and environment variations. They have done so by conducting four different experiments:

1. **Inertial Experiment**: A bottle half full of water was attached to the robot's fifth joint, altering its dynamics. IC and MPC controllers showed different offsets due to unmodeled dynamics, while all active inference controllers remained unaffected, the uAIC demonstrated higher control accuracy compared to the AIC.
2. **Elastic Constraint Experiment**: A rubber band was attached to the robot to change its dynamics drastically. Classic and uni-modal AIF controllers were significantly affected, unlike AIF based controllers.
3. **Human Disturbances Experiment**: Human operators pushed the robot in random directions to test compliance and recovery ability. MPC failed to recover, while all other controllers showed safe behavior and full recovery from disturbances.
4. **Noise Experiment**: Controllers were evaluated in the presence of proprioceptive noise. MAIC controllers demonstrated smoother behaviors due to multi-modal filtering, second best results were obtained by the uAIC, following after it the AIC.

2.2.3 The Model Reference Adaptive Controller

Opposed to the AIC and the uAIC, which are both model-free adaptive controllers, stands the MRAC, a model-based controller. Unlike model-free controllers that do not require a precise mathematical representation of the system they control, MRAC relies on an accurate description of the desired system's dynamics. This type of controller is fundamentally based on a reference model that defines the desired performance and behavior of the system.

The MRAC system continually adjusts itself based on the differences between the expected behavior (as defined by the reference model) and the actual behavior of the system being controlled. This process involves comparing real-time measurements from the actual system with the outputs predicted by the reference model. The discrepancies between these outputs are used to dynamically update the control parameters.

The key components of an MRAC system include the reference model, the adaptive mechanism, and the control law. The reference model specifies the ideal response of the system, serving as a benchmark for performance. The adaptive mechanism adjusts the controller parameters to minimize the error between the model's output and the system's output. Finally, the control law dictates how the control inputs should be modified based on the adaptive adjustments.

The main advantage of MRAC lies in its ability to handle system uncertainties and external disturbances effectively. By continuously adapting to changes, MRAC can maintain desired performance even in the presence of variations in system dynamics or external perturbations. This adaptability makes MRAC particularly suitable for applications where system parameters are not constant or fully known.

An example of the MRAC control diagram implemented in [13] is shown in Figure 2.8 This diagram illustrates the interaction between the reference model, the adaptive controller, and the plant. The reference model generates the desired output, which is compared with the actual output from the plant. The adaptive controller then modifies the control inputs to minimize the error between these outputs, thereby ensuring the system's behavior aligns with the reference model.

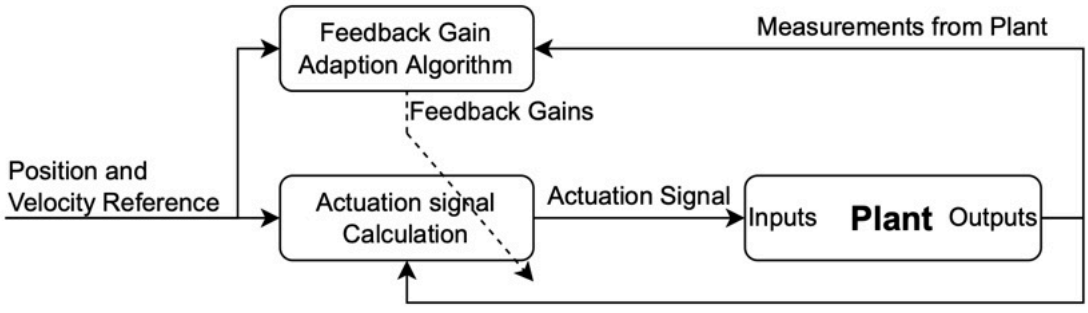


Figure 2.8: Control diagram of the MRAC [6]

3 System Overview

The following sections will provide an overview of both the hardware and software used in designing the system proposed in this project.

3.1 Robot Franka Emika

3.1.1 Hardware

For this project, a 7-DoF Franka Emika Panda Robot Arm is employed. The arm possesses a lifting capacity of 3kg and a reach extending up to 855mm [4]. Further details, specifically about the reachable area and joints limits are shown in Appendix A. The robot also features collaborative capabilities and an intuitive Graphical User Interface (GUI).

Additionally, the company offers two software libraries: *franka_ros* and *libfranka*. These libraries greatly facilitate the establishment of a functional connection to the robot via ROS and retrieving essential information via the dedicated Application Programming Interface (API), such as joint states or the position of the end-effector in world coordinates.
The Franka Emika Panda comes equipped with its own robotic gripper. However, it was decided not to utilize it and instead opt for a soft pneumatic gripper. This alternative gripper offers a compliant gripping force, making it suitable for handling delicate food products, as will be elaborated on later in this chapter.

Figure 3.1: 7-DOF Franka Emika Panda robot

3.1.2 Software

To control the robot, the three torque controllers characterized in Chapter 2.2 have been used. The MRAC and AIC from [13] and [14] were made already available at the beginning of this thesis project, whereas the uAIC has been implemented as part of it. These torque controllers leverage the two aforementioned libraries to communicate with the low-level controllers of the robot, the Franka Control Interface (FCI).

3.2 Desktop Computer

The project is developed using the desktop computer provided by the university lab.
The computer features an Intel(R) Core(TM) i7-7700 CPU running at 3.60GHz with 4 physical cores and 8MB of cache memory.
The preinstalled operating system is Ubuntu 20.04 64-bit and the code editor used is Visual Studio Code (VSCode).
The main programming languages used in the project are C++ and Python.

3.3 Gripper

The soft pneumatic robot gripper was provided by *The Gripper Company Aps* and chosen for its capabilities of offering a grasping force which is compliant with most delicate objects, such as fruit and veggies, and its ability to adapt to odd shapes.
It features four fingers reinforced by a metal core, and four control belts placed in between the fingers; these last are thin, elastic and loosely connected and they act as a passive dampening and controlling force [15].
The gripper only has two working positions: open and closed; the gripping forces are not declared by the producer. During a previous project course [7] there has been an attempt of applying some control on the opening and closing of the gripper, in order to be able to provide a varying grasping force. However, more costly equipment was necessary and thus the development of a control algorithm for the gripper was abandoned.
To allow the gripper to work some other equipment is necessary which will be briefly mentioned.

Figure 3.2: Soft robotic gripper from The Gripper Company

3.3.1 Complementary Hardware to Gripper

Together with the gripper also a pneumatic linear actuator and a 5/3 solenoid valve where provided.
A 5/3 solenoid valve is a type of pneumatic or hydraulic valve used to control the flow of a fluid, compressed air in this case, in a system. The ”5/3” designation refers to the number of ports and positions the valve has. The ports are:

- 1: inlet of pressurised air;
- 2: outlet of air when the first (right) solenoid valve is actuated;
- 4: outlet of air when the second (left) solenoid valve is actuated;
- 3,5: exhausts.

The working positions are:

- Neutral: the inflow of pressurised air is blocked. The exhaust are open;
- First actuated position: the air flows from 1 to 2 and back from 4 to 5;
- Second actuated position: the air flows from 1 to 4 and back from 2 to 3.

Moreover, in order to be able to control the solenoid valve (thus, the gripper) via ROS an Arduino UNO board is used, and a ROS node has been created and loaded to the board leveraging the *rosserial_arduino* library for Arduino.

3.4 Conveyor Belt

The conveyor belt was provided by the university laboratory and was used to emulate a dynamic e-grocery task where food is conveyed and picked by the robot to be sorted.

The blue rubber belt can travel both forward and backwards at a varying speed that can be controlled by a potentiometer. The potentiometer setting can range from a minimum of 0 to a maximum of 4000. The velocities associated with each setting of the potentiometer are detailed in Table 3.1. Due to physical limitations of the setup, which will be detailed in 4.1.2.2, the speed of the conveyor belt will be limited to that corresponding to a potentiometer setting of 2000.

Due to the lack of technical documentation, the speed associated with each potentiometer setting was found experimentally by measuring how long it would take for the conveyor belt to travel one meter. Twenty measurements were taken, yielding the results in Table 3.1.

Potentiometer Setting	Time to Travel 1 Meter (s)	Standard Deviation of 20 measurements	Speed (m/s)
500	44.44	0.075	0.0225
1000	22.598	0.080	0.0443
1500	15.015	0.066	0.0666
2000	11.304	0.122	0.0885
2500	9.001	0.044	0.1111

Table 3.1: Experimental measurements of the conveyor belt speed corresponding to different potentiometer settings.

3.5 Camera

Figure 3.3: Intel RealSense 435 depth camera

The implemented camera is the Intel RealSense Depth Camera D435, as shown in Figure 3.3.
It features two gray-scale cameras, a dots projector and an Red Green Blue (RGB) camera. The camera is connected to the computer through a 2-meter-long USB-C to USB-A cable.
To access the camera, the camera control software must be installed to obtain the device identification number. In addition, the Python library *pyrealsense* must be installed to work with the depth vision module.

3.6 Complete Setup

The complete setup employed for this project is shown in Figure 3.4.
The Franka Emika Panda robot is mounted onto a Mobile Industrial Robot (MIR) for purposes unrelated to this thesis project; therefore, its presence will be disregarded in all considerations.

The working area consists of a small wooden board mounted onto the MIR in front of the robot. The conveyor belt was selected for its compatible dimensions with the wooden board and its removable legs, which allow it to be placed at a height that enables comfortable robot movement. Previously, a different, larger conveyor belt was used, but it had to be disregarded due to its incompatible height and dimensions with the new working area.

The camera is mounted on a metallic frame built during a previous project. The frame is free to move and this can cause issues with object localization. To address this, a script is used to perform a camera-frame to robot-frame calibration at the start of each testing session.

As seen in Figure 3.4, the camera is positioned above the robot and the working area/conveyor belt, allowing the robot to move freely in between. This configuration, known as an eye-to-hand configuration, poses challenges for continuous object detection during task operation. For instance, when grasping a moving object on the conveyor belt, the robot can block the camera's Field of View (FOV), interrupting object detection and localization.

To mitigate this issue, a Kalman filter will be implemented to act as an object pose predictor, as later described in Chapter 4.1.2.2.

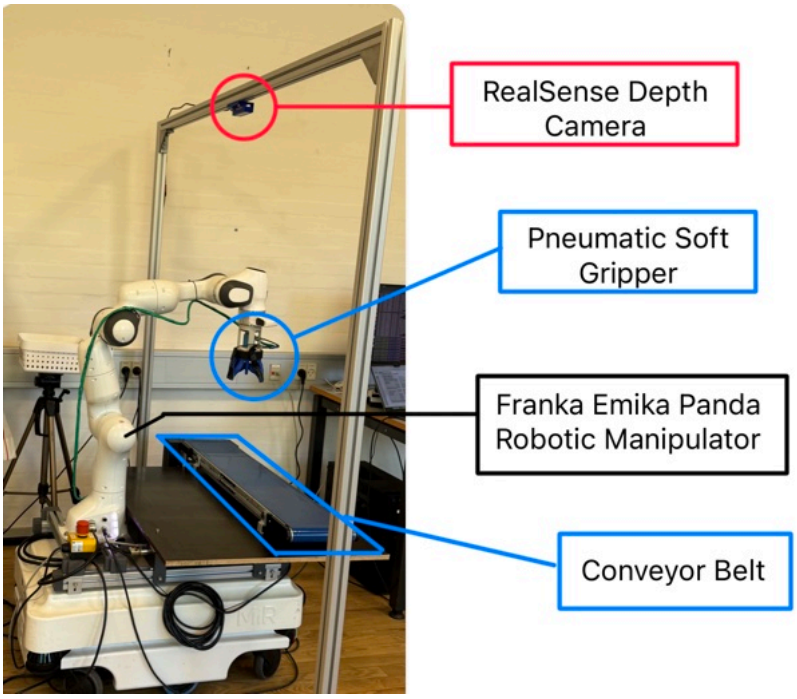


Figure 3.4: Complete setup used in the context of this thesis project. The main components described in this chapter have been highlighted for clarity.

4 Design and Implementation

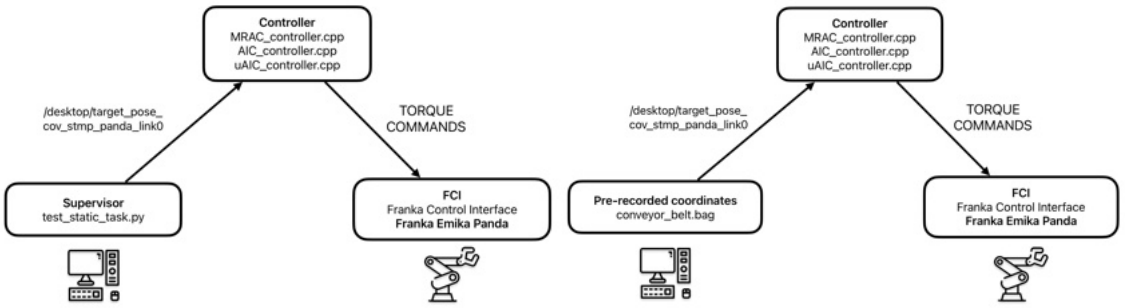
The ultimate objective of this project is to integrate the uAIC into the system and evaluate the performance of three distinct controllers in typical egrocery tasks. However, designing such a system is complex due to the need to integrate multiple components/nodes into a communication network. Therefore, the implementation strategy involves breaking down the process into milestones of increasing complexity, with each milestone introducing additional requirements to be met.
First, the two sets of tasks designed to assess the performance of the controllers will be introduced and detailed. Next, the implementation process of the uAIC in task-space will be described. Finally, the image processing required for object detection and localization in the e-grocery tasks will be explained.
The source code produced during the course of this thesis project can be found in [16].

4.1 Performance Evaluation Tasks

To effectively compare the performance of different controllers, it's essential to establish a set of Key Performance Indicator (KPI)s and define a series of tasks for testing each controller. Two sets of tasks have been defined, these will be referred to as "test tasks" and "e-grocery tasks", each comprising static and dynamic versions. Let's delve into the specifics of these tasks.

4.1.1 Test Tasks

The purpose of the test tasks is to analyze the performances of the controllers in terms of task-space tracking error, joint-space tracking error and torque used to accomplish each task.
For this set of tasks the system will not make use of the camera and the gripper, and the controllers will make the robot move to pre-defined coordinates.



(a) Static (b) Dynamic

Figure 4.1: Communication pipeline for the test tasks. In (a) the communication pipeline employed to navigate through static waypoints. In (b) the communication pipeline employed to track different trajectories.

4.1.1.1 Waypoints Navigation

In this initial task, the robot will be directed to move to specific predefined points in space. These coordinates will form a sequence simulating a typical pick-and-place operation, allowing for an assessment of the controllers' tracking performance in joint-space and along the three axes.

The Python script, *test_static_task.py*, publishes each set of coordinates to the controller node via the topic */desktop/target_pose_cov_stmp_panda_link0* for every waypoint (Figure 4.1a). The publication occurs at a constant rate of 300 Hz, which is the maximum publishing rate achievable by the script. Once the robot reaches a waypoint, it maintains that pose for ten seconds. This duration allows the controllers sufficient time to converge to the specified set-point, which is especially necessary during the implementation and testing phase of the uAIC.

4.1.1.2 Tracking of a Dynamic Reference

In this task, the robot is required to follow three specific trajectories. These trajectories simulate the hypothetical movement of an object traveling along a conveyor belt at varying speeds.

To create these trajectories, an object was placed on the conveyor belt and set into motion at different speeds. These speeds were achieved by adjusting the potentiometer setting of the conveyor belt to 1000, 1500, and 2000. Each of these potentiometer values corresponds to a velocity of the belt, which has been characterized previously in Chapter 3.4. The coordinates of the object's center were recorded into three separate rosbag files, each corresponding to a different trajectory.

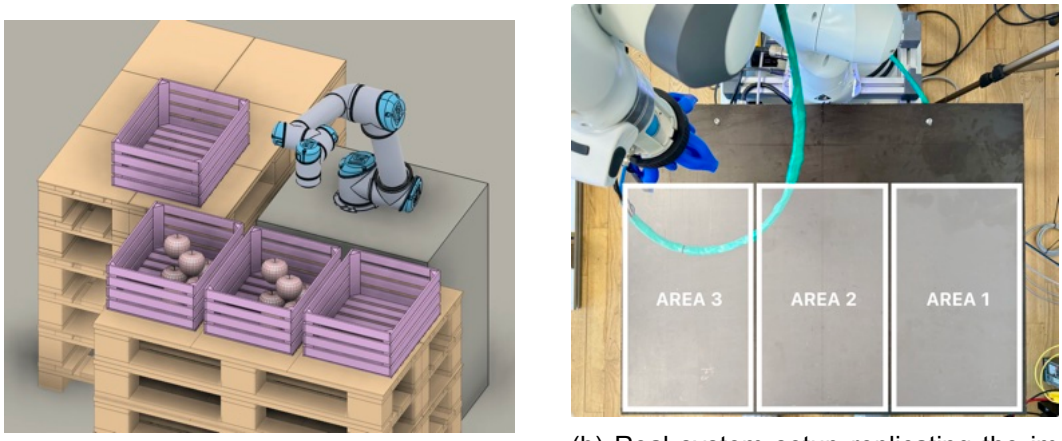
Subsequently, these bag files were employed to publish the object's coordinates to the controllers via the topic */desktop/target_pose_cov_stmp_panda_link0* (Figure 4.1b). This approach allowed observation of how fast and precisely the controllers could adapt to changing set-points, emulating real-world scenarios where objects might be conveyed at various speed.

4.1.2 Egrocery Tasks

The objective of the e-grocery tasks is to assess the performance of the controllers within a simulated e-grocery environment. In this set of tasks, the system will operate autonomously, utilizing both the camera and gripper. The following tasks have been chosen to reflect typical scenarios encountered in an e-grocery environment.

4.1.2.1 Grasping Static Objects From Unknown Locations

In the imagined e-grocery scenario, the robot is placed in front of different fruits or vegetables to be picked in order to be placed in a separate box.
To emulate this scenario, the object to be grabbed will be placed randomly on three predefined areas in front of the robot (Figure 4.2b). The system will be able to autonomously detect the object on the working area of the robot, enabling it to perform a typical pick and place task in complete autonomy. The overall communication pipeline for this task is illustrated in Figure 4.3.



(a) Imagined e-grocery scenario.

(b) Real system setup replicating the imagined scenario.

Figure 4.2: Comparison between the imagined static e-grocery scenario and its real system implementation.

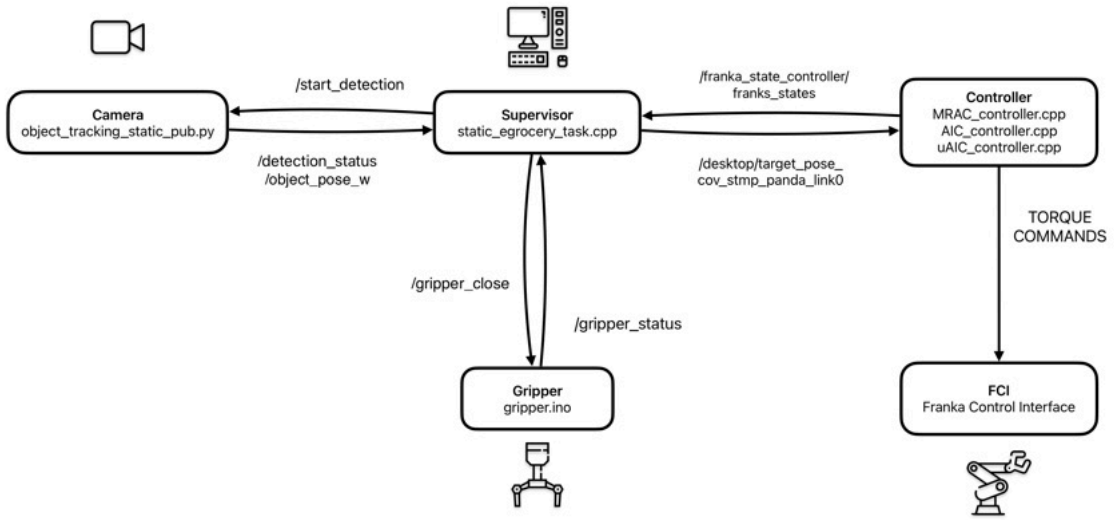


Figure 4.3: Static e-grocery task communication pipeline

Supervisor

The *Supervisor*, managed by *static_egrocery_task.cpp*, serves as the central node that communicates with and orchestrates all other nodes. This node implements the logic that can be seen in detail in Figure C.1.

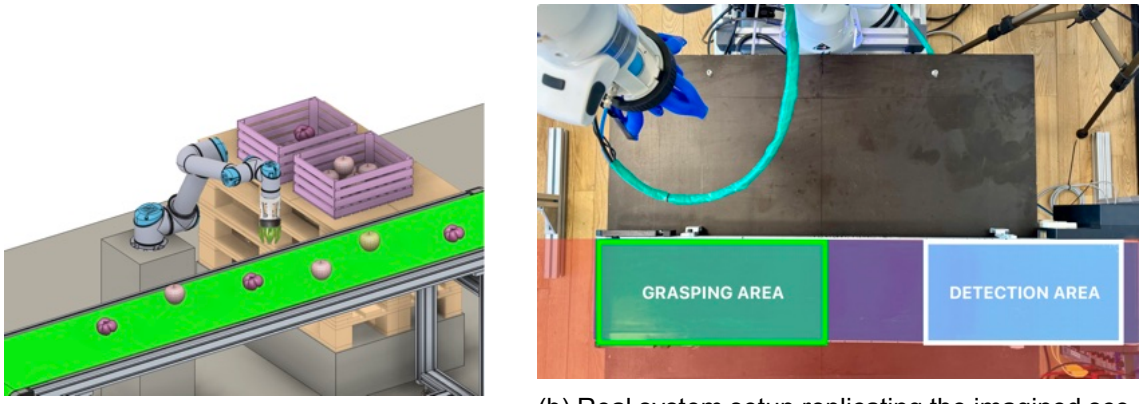
Camera

The camera functionality is handled by the *Camera* ROS node, operated by *object_tracking_static_pub.py*. This node initiates object detection upon receiving a command from the *Supervisor* via the */start_detection* topic. Once an object is detected, it forwards the detection status and its coordinates, in the robot frame, to the *Supervisor* via the */detection_status* and *object_pose_w* topics, respectively.

Gripper
The gripper is controlled by the *Gripper* node running on the Arduino UNO, leveraging the *rosserial_arduino* library. Commands to open or close the gripper are received through the */gripper_close* topic, with the action status forwarded on the topic */gripper_status* to the *Supervisor*.

Controller
Finally, the *Controller* node receives the desired end-effector pose via the */desktop/target_pose_cov_stmp_panda_link0* topic. Additionally, it publishes information about the status of the robot manipulator on the */franka_states_controller/franka_states* topic. Of great importance for this task is retrieving the end-effector pose relative to the base frame of the robot, this allows the *Supervisor* to verify whether and when the end-effector achieves the desired published pose.

4.1.2.2 Grasping Moving Objects From Unknown Locations



(a) Imagined e-grocery scenario.

(b) Real system setup replicating the imagined scenario.

Figure 4.4: Comparison between the imagined dynamic e-grocery scenario and its real system implementation.

In the imagined e-grocery scenario, the robot is positioned in front of a conveyor belt that transports fruits or vegetables to be picked and sorted into separate boxes based on criteria such as type or ripeness.

To emulate this scenario, the object to be grabbed will be conveyed by the conveyor belt at three different speeds, corresponding to potentiometer values of 1000, 1500, and 2000.
One of the limits of grabbing moving objects in an Eye-to-Hand camera-robot configuration is that the camera will not always be able to have a clear view of the object, especially when the robot manipulator obstructs the visual while moving on its working area and on the FOV of the camera.
To compensate for this, the system will integrate now the Kalman Filter, developed in [6], which will make prediction about the objects' pose at times when the camera does not have a view on it.

A second limit of this set-up is represented by the limited space available and length of the conveyor belt. In fact, in this set-up the camera is placed onto the operative area of

the robot, thus the available free space in which the camera can perform object detection is limited, as well as the area available to the robot to perform the grasping of the object. For this reason, the speed of the conveyor belt was limited to a potentiometer value of 2000. A higher speed would have made it impossible for the camera to collect a reasonable amount of data for accurate estimates and for the robot to grasp the object.

In an hypothetical real scenario, with a longer conveyor belt, the camera could be placed on another portion of the same and the robot could be free to move on the entire reaching area, thus increasing time available for detection and grasping.
The overall communication pipeline for this task is illustrated in Figure 4.5.

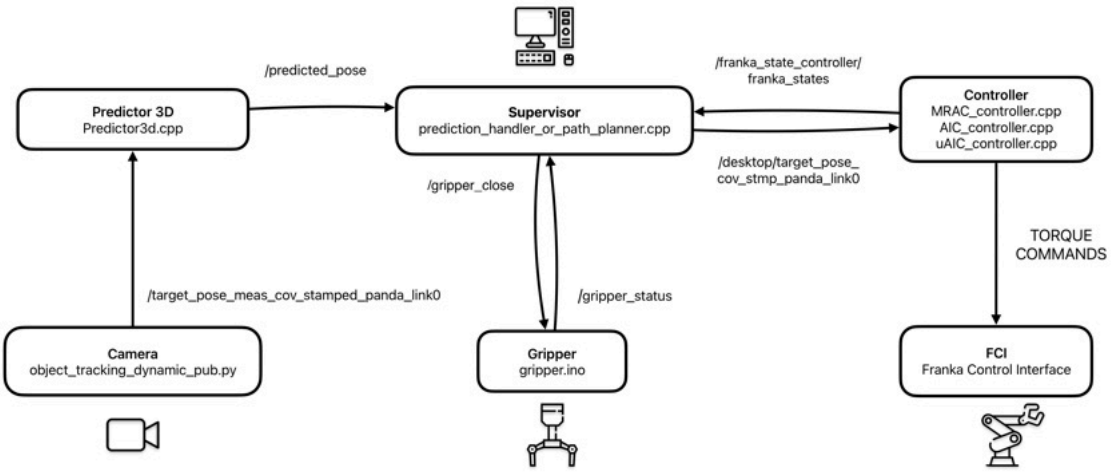


Figure 4.5: Dynamic e-grocery task communication pipeline

Supervisor
The *Supervisor*, is now managed by *prediction_handler_or_path_planner.cpp*, serves as the central node that communicates with and orchestrates all other nodes. This node implements the logic that can be seen in detail in Figure C.2.

Camera
The *Camera* ROS node is now operated by *object_tracking_dynamic_pub.py*. This node performs constant object detection and tracking on a specific area of the conveyor belt, as depicted in Figure 4.4b. While doing this, the node publishes the coordinates of the center-top of the object to the Kalman Filter via the topic */target_pose_meas_cov_stamped_panda_link*.

Predictor
The predictor is a Kalman Filter and, when provided with the 3D coordinates of an object for a certain period of time, it provides the n step ahead prediction of the pose of the object in the robot environment.
It was seen during the implementation of this task that the predictor had to be tuned in order to accommodate for the different performances provided by the various controllers. More specifically, the AIC controller required a 20-step ahead prediction, while the MRAC and uAIC requested a 15-step ahead prediction. The reasons behind this different configuration will be discussed in Chapter 6.

Gripper
The gripper is controlled by the *Gripper* node running on the Arduino UNO, leveraging the *rosserial_arduino* library. Commands to open or close the gripper are received through the */gripper_close* topic, with the action status forwarded on the topic */gripper_status* to the *Supervisor*.

Controller
Finally, the *Controller* node receives the desired end-effector pose via the */desktop/target_pose_cov_stmp_panda_link0* topic. Additionally, it publishes information about the status of the robot manipulator on the */franka_states_controller/franka_states* topic. Of great importance for this task is retrieving the end-effector pose relative to the base frame of the robot, this allows the *Supervisor* to verify whether and when the end-effector achieves the desired published pose.

4.2 Unbiased Active Inference Controller Implementation

The uAIC defined in 2.2.2 has been implemented on the Franka Emika Panda by [4] and the source code is available for use at [17].
Conceptually, similarly to the AIC, the uAIC consists of two parts.

- A state estimator which computes the position and velocity estimates $\hat{\theta}_0$, $\hat{\theta}_1$ and their derivatives $\hat{\theta}_0^1$, $\hat{\theta}_1^1$ which are each $n \times 1$ vectors. Where $n = 7$ and is equal to the number of DOF of the robot;
- An adaptive controller which computes the $n \times 1$ torque vector u based on difference between current angular position and velocity to the objective position and velocity, which are each $n \times 1$ vectors. In this project the objective joint velocity vector is always equal to zero, as it represent the desired final velocity of the robot once it has reached the desired joint position.

As it is mentioned by the same authors, the source code in [17] it is not ready to run and needs two things in order to work as intended:

- *franka_ros* or *libfranka* installed on the computer where the controller will be executed
- a high-frequency torque interface that is able to forward the torque commands to the joints. This is a ROS controller capable of handling torque commands.

If the franka_ros and *libfranka* were already installed on the laboratory computer, the torque interface needed to be implemented.
Furthermore, the provided controller only generates torque commands when provided with a desired joint position and velocity for each joint of the controller and thus only work in joint-space. To be able to work in task-space an IK resolver needs also to be implemented.
Seen the complexity of such implementation it was decided that the best course of action would have been to break it down into smaller milestones and at each new achievement move on with the following phase of the implementation.

All in all, the defined milestones have been the following:

- Implementation of the uAIC in joint-space;
- Implementation of the IK to allow the controller to work in task-space;
- Testing and debugging.

4.2.1 Implementation of the uAIC in Joint-Space

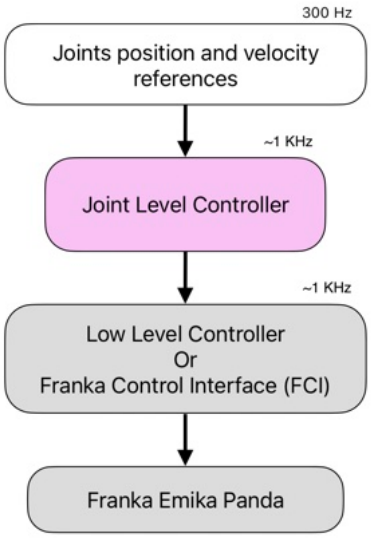


Figure 4.6: Communication pipeline for the uAIC working in joint-space. In white the Supervisor node, in pink the Controller node, and in gray the FCI and Franka Emika Panda. The publishing rates for each ROS node are indicated at the top of each box.

As previously mentioned, enabling the uAIC from [17] to control the robot requires implementing a high-frequency torque interface that sends the evaluated necessary torques to the robot's low-level controllers. This can be achieved using a ROS effort controller, for example.

To facilitate the implementation process, given the complexity and potential time required, it was decided to utilize the existing infrastructure used for the AIC in [3], with the source code available at [14]. However, the two controllers used different libraries and methods for publishing torques to the robot's low-level controllers. Therefore, the uAIC code needed to be adapted to use the *libfranka* library employed in the AIC. After making the necessary adaptations to the uAIC code, it was implemented within the existing ROS infrastructure of the AIC. The communication pipeline for this setup is illustrated in Figure 4.6.

The obtained controller worked in three phases:

- **Init:** this is executing once when the controller is launched. Here, all precision matrices, controller constants, variances of beliefs and sensory inputs are initialised;
- **Starting:** also this is executed only once. Here, ROS publishers and subscribers are initialised as well as the belief about joints position and velocity set equals to the gathered sensory data;
- **Update:** this is executed repeatedly at a rate of approximately 1 KHz. In this phase the state estimator and controller continuously evaluate the states estimation and control torques.

To verify the correct implementation of the controller, it was tested by passing joint position references and observing the response. The communication pipeline for this setup is illustrated in Figure 4.6 and the results shown in 5.2.

4.2.2 Implementation of the uAIC in Task-Space

With the controller now functioning properly in joint-space, the next objective is to enable it to receive references in Cartesian space by implementing an IK resolver.
At the beginning of this project, an inverse kinematics resolver was already made available by the work carried out by student Christian Kampp Kruuse, as documented in [6].

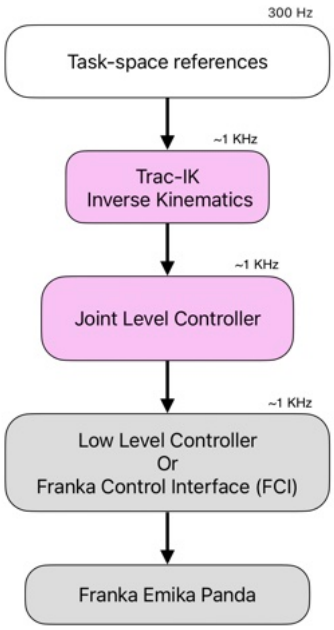


Figure 4.7: Communication pipeline for the uAIC working in task-space. In white the Supervisor node, in pink the Controller node, and in gray the FCI and Franka Emika Panda. The publishing rates for each ROS node are indicated at the top of each box.

As depicted in Figure 4.7, the supervisor node (in white) is now transmitting task-space references to the controller node (in pink), allowing for a more intuitive control of the robot within the performance assessment tasks and e-grocery tasks.
The conversion of task-space references into joint-space references is executed by Trac-IK, which subsequently enables the controller to generate torque controls. These torque controls are then transmitted to the FCI, and ultimately to the robotic arm.

After undergoing testing and debugging, the controller showed occasional issues, particularly evident in certain poses. Notably, the first three joints, being the most powerful, exhibited significant jittering, resulting in pronounced instability. Additionally, these fluctuations occasionally triggered the error message: *power_limit_violation*, preventing the robot from moving and continuing the control loop.

This error is triggered by the *libfranka* library if the ”maximum allowed power” is reached [18]. However, neither the official website of Franka Emika nor the Franka Emika Community forum stated what exactly this error code meant and from which joint limit violation it was caused.

From the data gathered while the error occurred, it was assumed that it was caused by a violation of the maximum speed allowed for the joints (Figure 4.8). The joint limits can be seen in Table A.1.

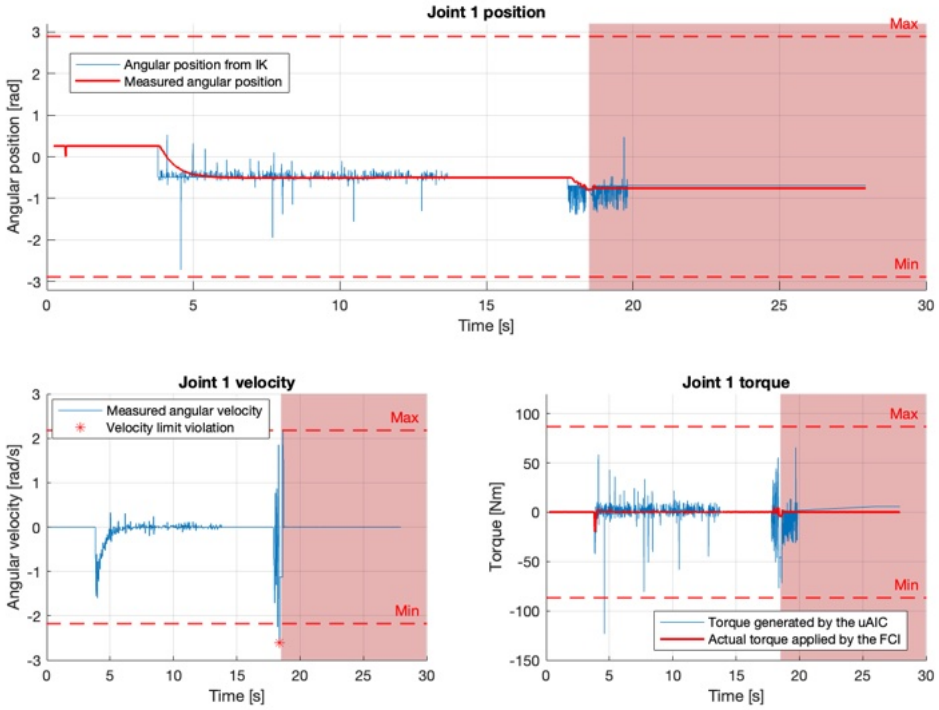


Figure 4.8: Evolution over time of joints position, velocity and torque for the first joint.

In this test, the robot was commanded to move the end-effector to positions increasingly distant from the base. Specifically, the end-effector had to move to $x = 0m, \quad y = -0.5m, \quad z = 0.5m$ and hold this position for fifteen seconds. Then, it had to move to $x = 0m, \quad y = -0.6m, \quad z = 0.5m$ and hold this new position for another fifteen seconds. However, it was observed that immediately after reaching the second commanded pose, the robot stopped (indicated by the red area in the graphs) and communicated the ”maximum allowed power” error.

Figure 4.8 illustrates the behavior of the first joint during this brief test. The three plots in the figure represent the evolution over time of:

- The angular position generated by the IK and the measured angular position (top plot);
- The measured angular velocity (bottom left plot);
- The torque signal generated by the uAIC and the actual torque generated by the FCI and sent to the robot (bottom right plot).

In the first plot, it is noticeable that the position reference generated by the IK is highly noisy with significant spikes. This occurs because the IK, at each iteration, evaluates different optimal solutions to the IK problem and does not maintain a consistent solution to the same problem. Consequently, since the desired angular position is directly used to evaluate the desired torque, this torque is also affected by the noise.

In the third plot, it is evident that the generated torque differs from the one generated by the controller. This discrepancy arises because the *libfranka* library first applies a filter

to the incoming desired torque to avoid abrupt changes in the torque applied to the joints. Additionally, it corrects the torque to account for motor friction and the gravity of the entire kinematic chain [18]. Therefore, even though the uAIC evaluates a noisy torque that exceeds the maximum allowed torque for joint 1, the robot does not stop because the actual torque sent to the Panda robot remains within the allowed range and stays low throughout the test, albeit with some oscillations.

These torque oscillations inevitably cause oscillations in the joint velocity (bottom left plot). At around time $t = 18.7s$, the joint velocity exceeds the maximum allowed speed of $2.1750\frac{rad}{s}$. It is suspected that the *power_limit_violation* error is triggered whenever the velocity of a joint exceeds the allowed limits.

At this point, a copy of the controller was also implemented in Matlab and tested on real data from the test but with different joint position references. The Matlab simulations confirmed the findings from the controller working directly in joint-space: a noisy reference causes the uAIC to generate noisy torque commands, while a well-behaved joint reference results in well-behaved torque commands.

Considering this, the problem could be resolved with a consistent joint-reference position. The options were to either find an inverse kinematics solver that provides a unique solution over time for the same task-space reference or to filter the references generated by Trac-IK. The first option was deemed time-consuming given the research and implementation required.
Despite this, an attempt was made to implement the IK solver from [19], which involved importing the C++ class and calling the provided function. This IK solver included a ”case-consistent” function that iteratively returned a solution close to the previously provided joint configuration.

Although this approach seemed optimal in theory, the algorithm did not check for self-collision and occasionally provided solutions that violated the robot’s joint limits. Consequently, it was decided to choose the second option and implement a low-pass filter for the references provided by Trac-IK.

4.2.2.1 Filtering of the Trac-IK Reference

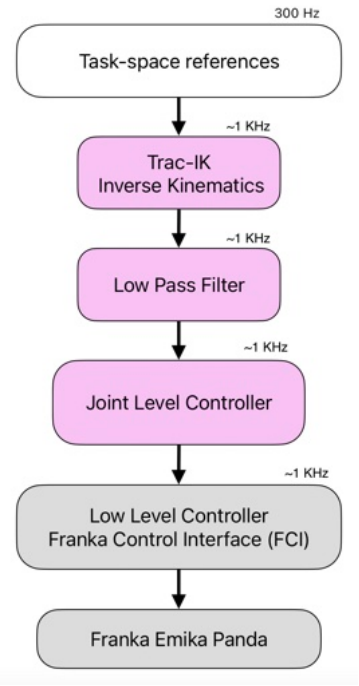


Figure 4.9: Communication pipeline for the uAIC working in task-space with the filtered Trac-IK Reference. In ☐ white the Supervisor node, in ☐ pink the Controller node, and in ☐ gray the FCI and Franka Emika Panda. The publishing rates for each ROS node are indicated at the top of each box.

With the uAIC implemented in Matlab it has been easy to also test the application of a low-pass filter applied to the Trac-IK reference.
The IK reference is generated at the same frequency of the *update* phase of the control loop, thus it was computed at frequency of $997.9 \quad MHz$. To remove very high and fast changes from the signal, a cut-off frequency of $f_c = 3.142 \quad MHz$ was chosen and the following low-pass filter transfer function was obtained:

$$H(s) = \frac{\omega_0}{s + \omega_0} = \frac{3.142}{s + 3.142} \tag{4.1}$$

And with the Matlab function *c2d* the following discrete transfer function was obtained:

$$H(z) = \frac{0.001573z + 0.001573}{z + 0.9969} \tag{4.2}$$

The results yielded by the application of the low-pass filter to the IK reference can be seen in Chapter 5.1.

4.2.2.2 Tuning of the uAIC

During the implementation phase, after the gripper was mounted onto the robot, it was noticed that the robot could not converge to the generated reference. For this reason, the uAIC underwent re-tuning of its control parameters. Due to time constraints, this tuning was carried out using a trial-and-error method. Table 4.1 lists the original tuning parameters of the controller, while the new parameters can be seen in Table 4.2. A discussion about the tuning method of the uAIC and its respective drawbacks is further elaborated in 6.2.

	Joint Number						
	1	2	3	4	5	6	7
SigmaP_mu	1/15.0	1/15.0	1/15.0	1/15.0	1/15.0	1/15.0	1/15.0
SigmaP_muprime	1/30.0	1/30.0	1/30.0	1/30.0	1/30.0	1/30.0	1/30.0
SigmaP_yq0	1	1	1	1	0.1	0.05	0.01
SigmaP_yq1	1	1	1	1	0.1	0.05	0.01
k_mu	51.6	51.6	51.6	51.6	51.6	51.6	51.6
k_a	200	200	200	200	200	200	200
k_p	55	35	35	15	15	12	2
k_i	0.01	0.01	0.01	0.01	0.01	0.01	0.01
k_d	30	30	30	30	10	30	30

Table 4.1: Table of the original tuning parameters for the uAIC as in [17]

	Joint Number						
	1	2	3	4	5	6	7
SigmaP_mu	1/15.0	1/15.0	1/15.0	1/15.0	1/15.0	1/15.0	1/15.0
SigmaP_muprime	1/30.0	1/30.0	1/30.0	1/30.0	1/30.0	1/30.0	1/30.0
SigmaP_yq0	1	1	1	1	0.1	0.05	0.01
SigmaP_yq1	1	1	1	1	0.1	0.05	0.01
k_mu	51.6	51.6	51.6	51.6	51.6	51.6	51.6
k_a	200	200	200	200	200	200	200
k_p	55	50	40	35	15	15	4
k_i	0.01	0.0138	0.009	0.013	0.011	0.0068	0.003
k_d	30	17	14.5	9	9	13	8

Table 4.2: Table of the final tuning parameters used for the uAIC in this project

4.3 Image Processing

The integration of a camera into the project was done with the sole purpose of performing object detection, tracking, and localization in the e-grocery task.

Object detection involves identifying multiple objects within an image or a video frame. The goal is to provide bounding boxes around them, indicating their location within the image.

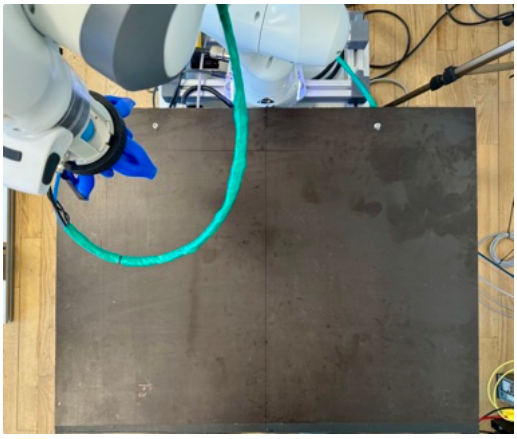
Object tracking focuses on following the movement of a specific object or multiple objects across consecutive frames in a video. The primary objective is to maintain the identity of the objects throughout the video sequence. This will be particularly relevant during the execution of the e-grocery task as defined in 4.1.2.

Localization refers to the process of determining the precise spatial location of an object within an image or a scene. In this context, the location in the camera frame will be determined for the center point of the bounding boxes, which will represent the center top part of the object.

4.3.1 Object Detection and Tracking

In order to locate objects within the working area, the image captured by the camera needs to be processed. In this scenario, only the RGB module is utilized, while the other two gray-scale cameras are employed for depth-vision purposes. Although the RGB module can capture images at a resolution of 1920 by 1080 pixels, the image quality has been intentionally reduced to match the resolution of the gray-scale cameras, which operate at 1280 by 720 pixels. This adjustment ensures accurate mapping of every pixel from the RGB camera onto the gray-scale ones, thereby easing object localization in 3D space.

Positioned at 1.32 meters above the base frame of the robot, the camera also captures data from areas that are not of interest for the task. For this reason, a mask has been applied to reduce the detection area to include only the area of interest: the black table for the static task and a small portion of the conveyor belt for the e-grocery task.



(a) Unfiltered

(b) Unfiltered

(c) Filtered

Figure 4.10: Entire image captured from the camera (a). Mask applied for grasping static objects (b). Mask applied for grasping moving objects (c).

The captured image is then converted into a black and white gradient format to facilitate contrast analysis. Object detection relies on contrasting the object against the underlying surface. By identifying pixel curves where the contrast exceeds a predetermined threshold, the object's area is delineated.

A filtering process eliminates detected objects below a certain area threshold, thereby mitigating false positives caused by areas of higher brightness, such as lamp reflections.

Once the contour curve defining the object's boundaries is found, its center is determined by averaging the sum of x and y pixel coordinates along the contour.

After identifying the center, it is continuously tracked across frames and given a unique ID. This is achieved by comparing the Euclidean distance of each detected center to its position in the previous frame. If this distance is less than 25 pixels, it is considered part of the same object.

Subsequently, transforming these pixel coordinates into world coordinates will allow for object localization in the physical environment.

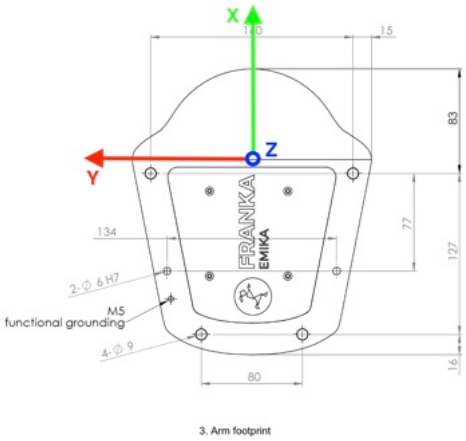
4.3.2 Object Localization

To perform object localization, the pinhole camera model is employed.

The pinhole camera model describes the mathematical relationship between the coordinates of a point in three-dimensional space and its projection onto the image plane of an ideal pinhole camera. In this model, the camera aperture is represented as a single point, and no lenses are used to focus light. This model will be used to determine the pose of an object in three-dimensional space from its pixel coordinates on the image plane.

To achieve this, the intrinsic and extrinsic camera parameters are required.

4.3.2.1 Extrinsic Camera Parameters



(a) Robot base frame

(b) Camera frame

The extrinsic camera parameters pertain to the camera's position and orientation relative to a world frame, independent of the camera's lens and sensor properties. These parameters are typically determined through a manual assessment of the camera's pose concerning the world frame. The extrinsic camera parameters are:

- Elevation of 1.320 m along the z axis of the robot frame;
- Translation of 0.355 m along the x axis of the robot frame;
- Translation of 0.0 m along the y axis of the robot frame;
- Rotation of 180 degrees along the y axis of the robot frame;
- Rotation of 90 degrees along the z axis of the robot frame;

4.3.2.2 Intrinsic Camera Parameters

On the other hand, intrinsic camera parameters are contingent on the camera's specific lenses and sensor. They encompass factors like focal length, aperture, field-of-view, and resolution, which collectively define the intrinsic matrix of the camera model. To obtain these parameters, a camera calibration process is essential.

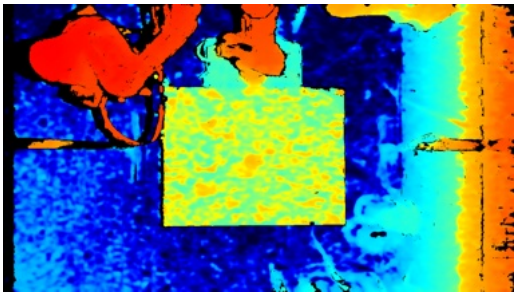
The camera calibration process has been conducted using the $camera\_calibration.py$ script, available in the project's GitHub repository. This script yields the intrinsic matrix of the camera, providing details such as the focal length in pixels (f_x, f_y) and the coordinates of the principal point (c_x, c_y). The principal point represent the point of intersection between the rays emanating from the pinhole and the image plane. In simpler terms, it can also be seen as the origin point of the camera frame.

The calibration is brought out by capturing images of a checked board at different angles and poses; the script needs to know the number of squares on the check board and needs to be provided with a certain number of captures of the same. The result of such calibration is the following intrinsic parameters matrix:

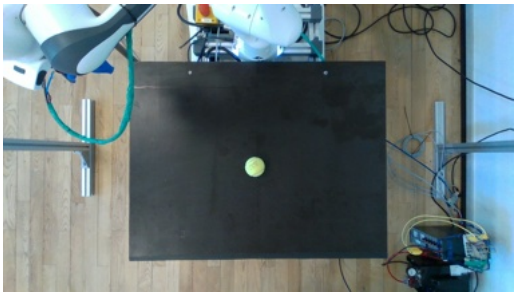
$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 913.77 & 0 & 640 \\ 0 & 910.35 & 360 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

4.3.2.3 Depth-Data Extraction

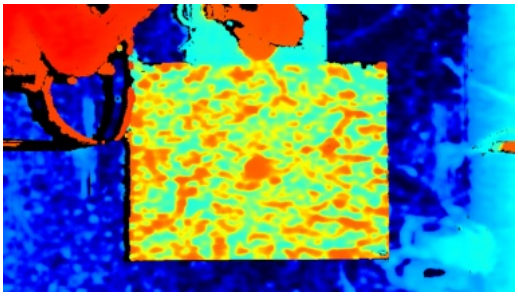
To enable autonomous object localization in 3D space, depth vision provided by the camera module is utilized. This is achieved through the *pyrealsense* library, supplied by Intel. The library grants access to the depth data stream produced by the camera's built-in chip and offers a range of filters that enhance the accuracy of distance measurements from the depth data. The depth vision of the camera module uses two black and white cameras, which have a different FOV compared to the RGB module, as shown in Figure 4.12 (a) and (b).



(a) Non aligned depth-frame



(b) Color-frame



(c) Aligned depth-frame

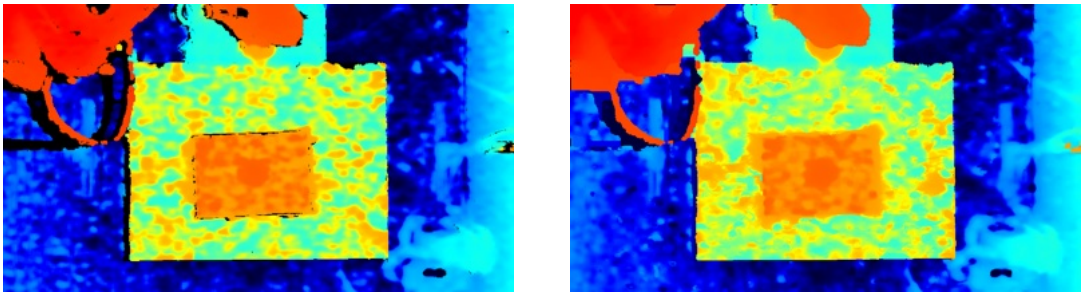
Figure 4.12: Depth-frame and color-frame misalignment

The consequence of this is that that pixel coordinates on the depth frame correspond to different points in space compared to the same pixel coordinates on the color frame. To address this issue, the $pyrealsense$ library offers a function to align the depth frame and the color frame. In Figure 4.12c

The filter applied to the data stream are the following, in the following order, as suggested in [20]:

- **Spatial Edge-Preserving Filters.** The filter performs a series of 1D horizontal and vertical passes or iterations, to enhance the smoothness of the reconstructed data.
- **Temporal Filter.** The temporal filter is intended to improve the depth data persistency by manipulating per-pixel depth values based on previous frames.
- **Holes Filling Filter.** As the name suggests, this filter implements several methods to rectify missing data in the resulting image. The missing data has been filled by assigning a value from the neighboring pixel which is furthest away from the sensor.

A detailed examination of the theoretical principles behind the operation of each filter has been omitted as it was considered beyond the scope of this thesis's objectives. Filters were selected and applied based on their observed positive effects on depth measurements.
In Figure 4.13 it is possible to appreciate a before and after the filters have been applied on the depth-data.



(a) Unfiltered depth-image (b) Filtered depth-image

Figure 4.13: Effects of applying post-processing filters on depth-data

4.3.2.4 From Pixel Coordinates to Camera Frame Coordinates

Found both the intrinsic and extrinsic camera parameters, it is possible to continue with the calculation to transform any coordinate found with respect to the camera coordinates frame to a set of coordinates into the robot coordinate frame. To do so, it is possible to use the following relations:

$$r = -f_x \frac{x}{z} + c_x \qquad c = -f_y \frac{y}{z} + c_y \tag{4.4}$$

Where, f_x, f_y, c_x and c_y are defined in equation 4.3.2.2, z is obtained by the depth reading and represents the distance of the objects from the camera plane, as detailed in the previous paragraph.
Lastly, r and c are the pixel coordinate of the object, which equals to x and y coordinates

on the image frame. Thus, with all these parameters known it is possible to obtain the coordinates of an object in the camera frame by leveraging Equation 4.4.

$$x = (c_x - r)\frac{z}{f_x} \qquad y = (c_y - c)\frac{z}{f_y} \tag{4.5}$$

4.3.2.5 Transformation From Camera Frame to Robot Frame

Once the coordinates of the object in the camera frame are determined, they can be mapped to coordinates in the world frame (or robot frame).
To do so, a homogeneous transformation matrix of the following kind need to be defined:

$$H_c^w = \begin{bmatrix} R_c^w & o_c^w \\ 0_{1x3} & 1 \end{bmatrix} \tag{4.6}$$

The matrix above describes the translation and rotation of the camera frame with respect to the robot frame. R_c^w is the Rotation matrix that will be defined through composition of rotations and o_w^c is the translation of the camera frame with respect to the world frame.
The camera frame is obtained by rotating it $\theta = 180$ degrees about the world y-axis, followed by a $\phi = -90$ degrees rotation about the world z-axis. Following the principle of pre-multiplication, the resulting rotation matrix is:

$$R_c^w = R_{y,\theta}R_{z,\phi} = \begin{bmatrix} cos(\phi) & -sin(\phi) & 0 \\ sin(\phi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{4.7}$$

While the translation o_w^c is defined by the extrinsic camera parameters, thus:

$$o_c^w = \begin{bmatrix} 0.355 \\ 0.0 \\ 1.320 \end{bmatrix} \tag{4.8}$$

Finally, the homogeneous transformation matrix in 4.6 becomes:

$$H_c^w = \begin{bmatrix} 0 & -1 & 0 & 0.355 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1.320 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

Whit this homogeneous transformation matrix any set of coordinates defined in the camera frame can be transformed into the world frame by using the relation:

$$P_w = H_c^w P_c \tag{4.10}$$

Where P_c are the coordinates of an object in the camera frame, and P_c are the resulting object coordinates in the world frame.

5 Results

This chapter provides an overview of the results obtained from the experiments conducted during the course of this project. Given the number of metrics and controllers used, a large amount of data has been generated from the laboratory experiments. Therefore, only the most relevant time series plots and tables will be presented in this chapter. For more in depth data the reader will be referred to specific section of the Appendix.

5.1 Filtering of the Trac-IK Reference

In Figure 5.1 the effect of applying the filter to the Trac-IK reference can be appreciated.

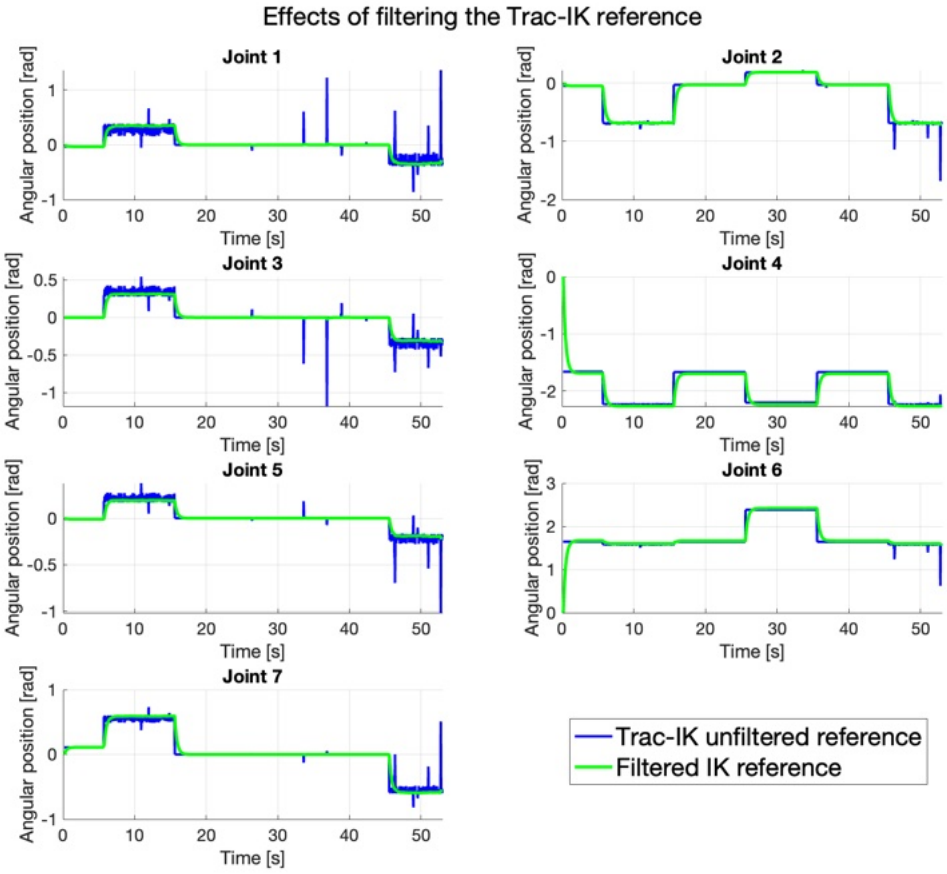


Figure 5.1: Effects of applying a low-pass filter to the Trac-IK reference. For each joint, the angular position reference as provided by the Trac-IK (in — blue) is compared against the filtered reference (in — green).

5.2 Tuning of the uAIC

This section reports the results obtained from the tuning of the uAIC.
First, Figures 5.2 and 5.3 illustrate the effect on tracking a reference in joint-space, with the uAIC tuned using the original tuning parameters, before and after mounting the gripper to the robot.
Then, Figures 5.4 and 5.5 illustrate the effect on tracking a reference in joint-space, with the uAIC tuned using new tuning parameters, before and after mounting the gripper to the robot.

For further details about the tracking performance in task-space, as well as joint-space and task-space tracking errors the reader is referred to Appendix D.

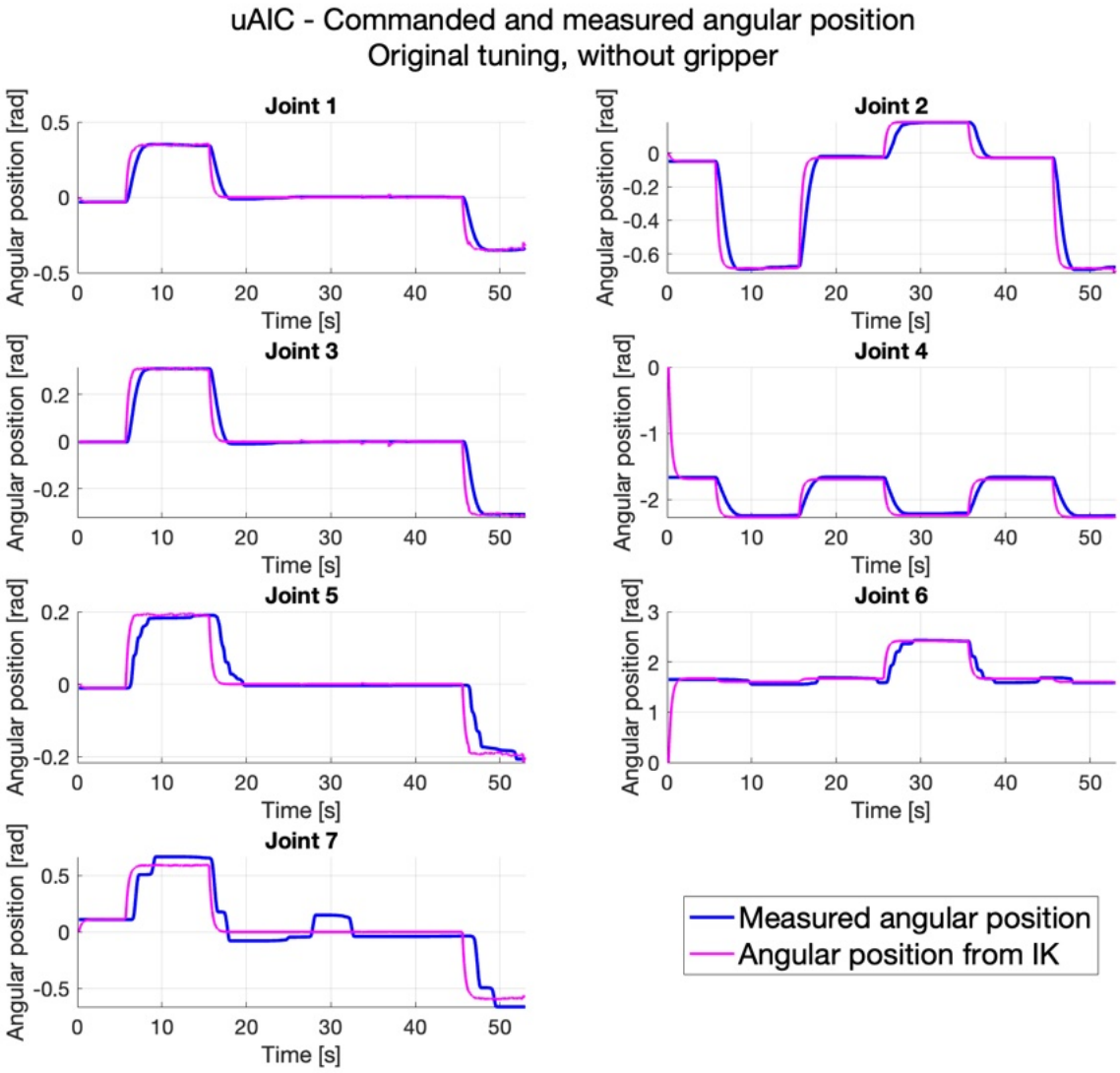


Figure 5.2: Joint-space tracking of the uAIC when navigating through static waypoints without the gripper attached to the robot. Here the uAIC is tuned with the original parameters as in [4]. For each joint, the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

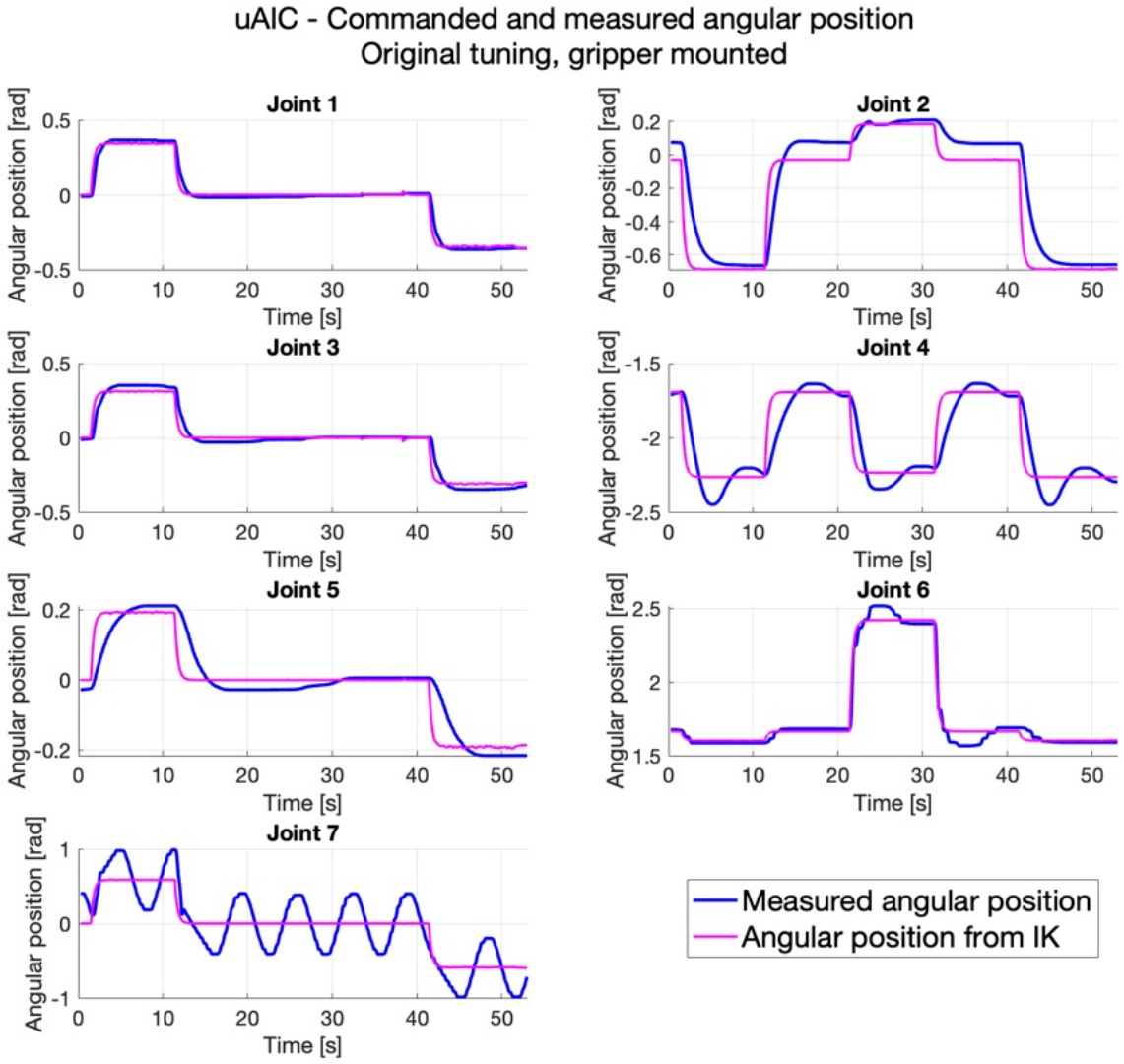


Figure 5.3: Joint-space tracking of the uAIC when navigating through static waypoints with the gripper attached to the robot. Here the uAIC is tuned with the original parameters as in [4]. For each joint, the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

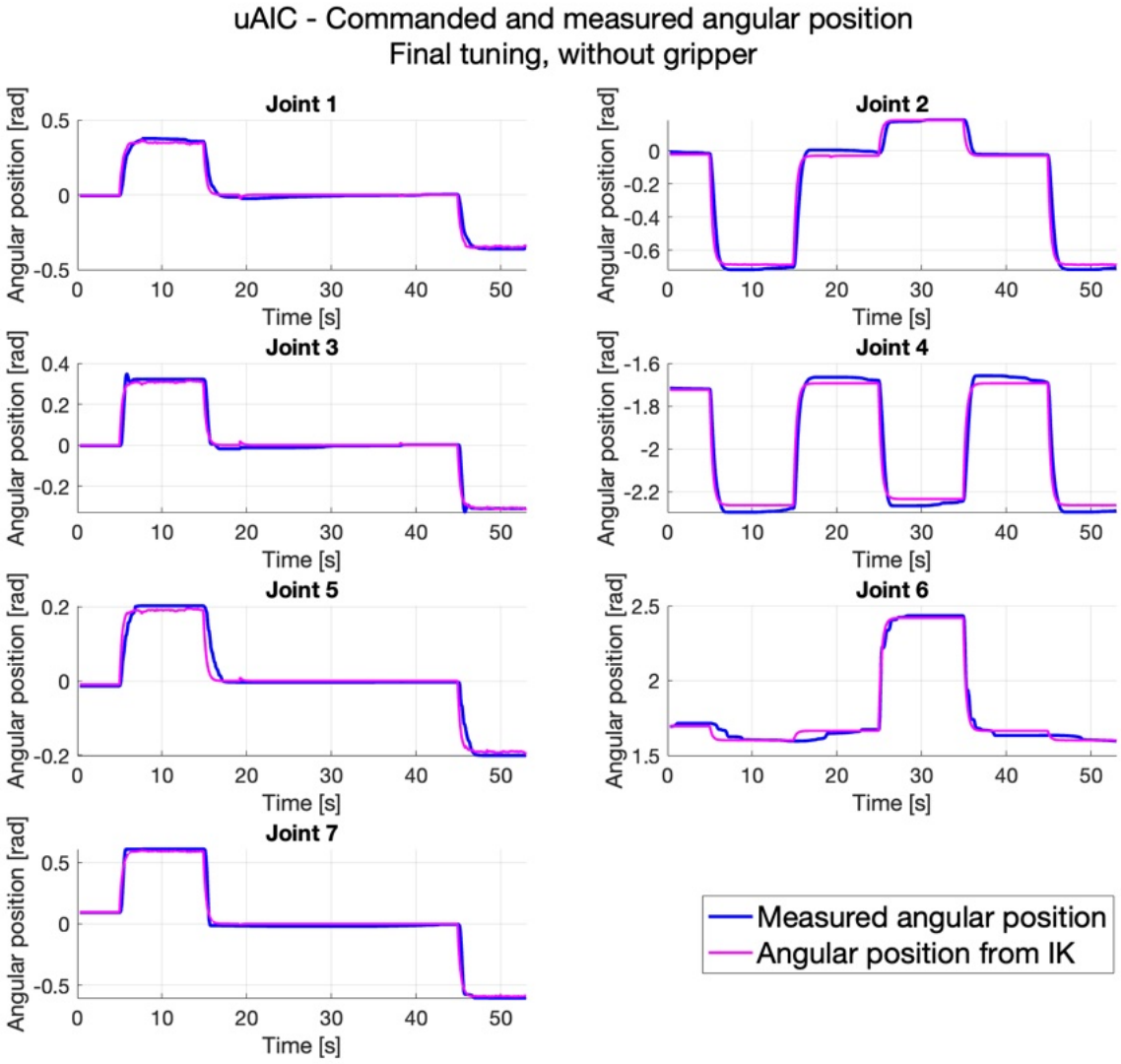


Figure 5.4: Joint-space tracking of the uAIC when navigating through static waypoints without the gripper attached to the robot. Here the uAIC is tuned with the parameters in Table 4.2. For each joint, the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

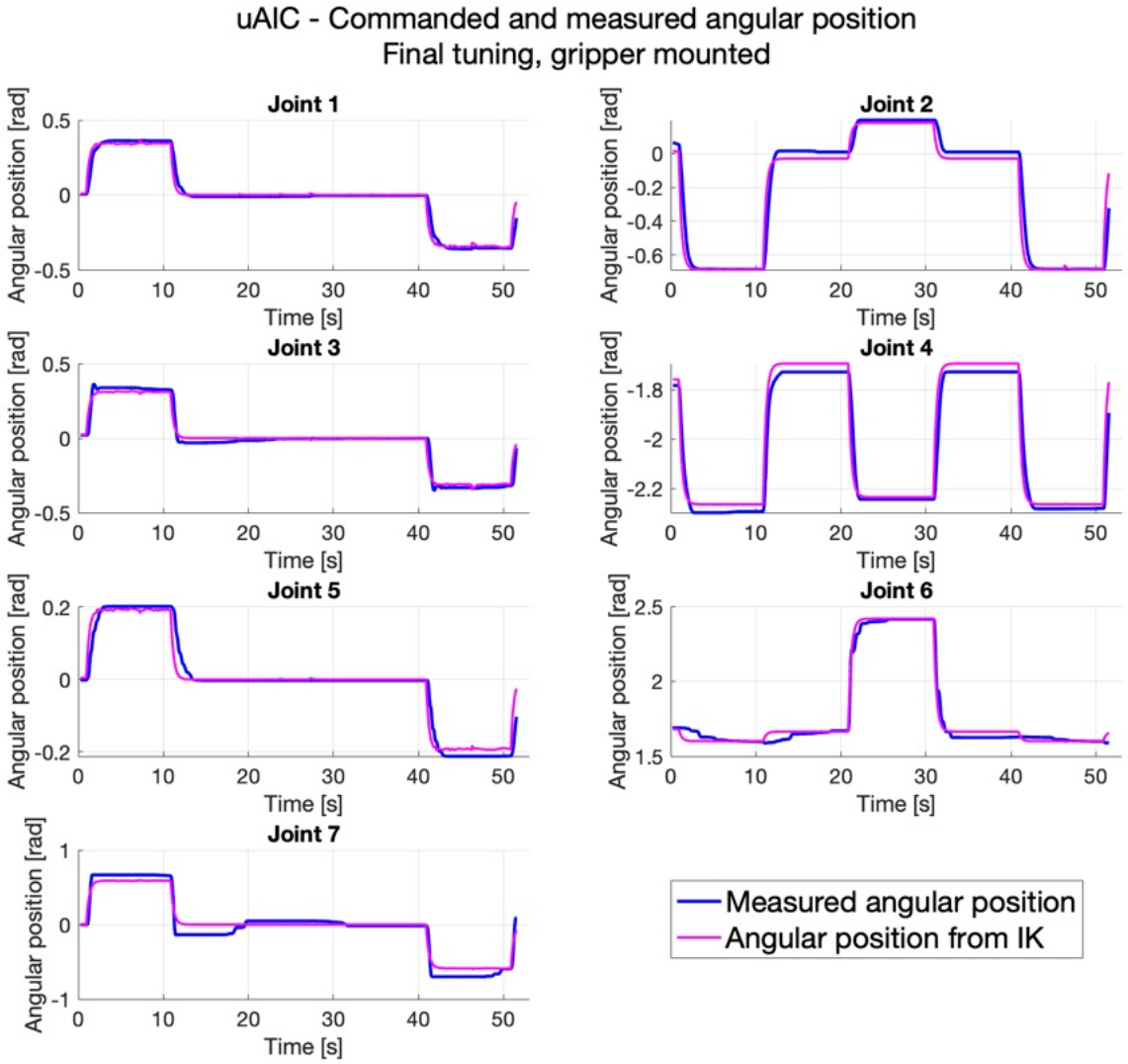


Figure 5.5: Joint-space tracking of the uAIC when navigating through static waypoints with the gripper attached to the robot. Here the uAIC is tuned with the parameters in Table 4.2. For each joint, the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

5.3 Depth Camera Distance Measurement Results

In this section the results obtained by implementing the depth-vision, as described in 4.3.2.3 are briefly introduced.
In these experiments, the z-coordinate have been recorded six times over the course of thirty seconds using both unfiltered and filtered depth-data

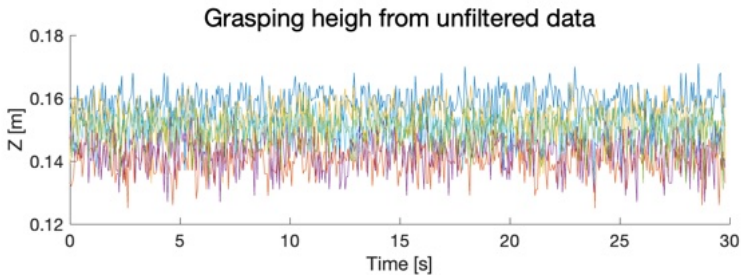


Figure 5.6: Time series plot showing six 30-second measurements of the z-coordinate of a detected object using unfiltered depth data.

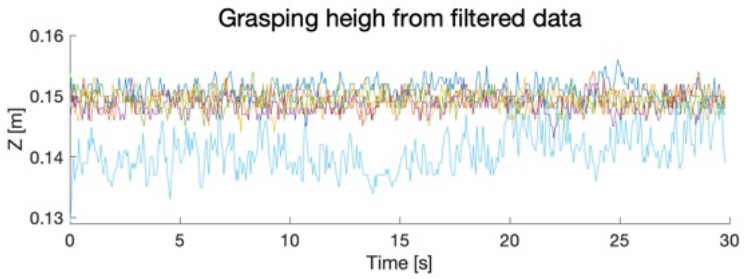


Figure 5.7: Time series plot showing six 30-second measurements of the z-coordinate of a detected object using filtered depth data.

Overall, applying the filter successfully reduced the standard deviation from the mean value.
In particular, the standard deviation for the measurements conducted with the unfiltered depth data reported the following values:

$$[0.0045 \quad 0.0046 \quad 0.0061 \quad 0.0059 \quad 0.0052 \quad 0.0046] \quad m \tag{5.1}$$

Whereas, the measurements conducted with the filtered depth-data reported the following standard deviation values:

$$[0.0016 \quad 0.0015 \quad 0.0015 \quad 0.0015 \quad 0.0015 \quad 0.0034] \quad m \tag{5.2}$$

The measurement of the X and Y coordinates of the pose of the object were also recorded, but their measurement were not affected by the application of the filters. On average the standard deviations on the X and Y measurements are:

$$\text{standard deviation of } X = 4.303e-04$$
$$\text{standard deviation of } Y = 3.331e-04$$

5.4 Comparison of Tracking Performances

In this section, the results obtained from the experiments described in 4.1.1.1 and 4.1.1.2 are presented.
As mentioned in section 1.3, the performance metrics chosen to evaluate the performances of the three controllers on the test tasks are:

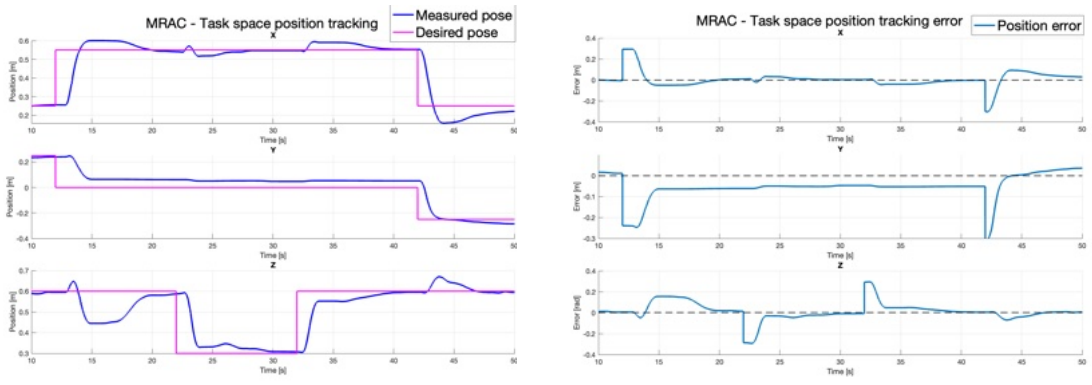
- Tracking error in task-space;
- Tracking error in join-space;
- Commanded torque.

For both experiments, time series plots of the tracking performance in task-space, along with their respective errors, will be shown. Additionally, a table reporting the Root Mean Square (RMS) of the tracking errors (in both task-space and joint-space) and the utilized torques will be included; these tables will be crucial in analyzing controller performance and discussing the results.

5.4.1 Waypoints Navigation

The following results were obtained on the tests described in 4.1.1.1.
For further time series plots of the joint-space tracking performance and the torque usage for all seven joints of the robot can be found in Appendix E.1.1.

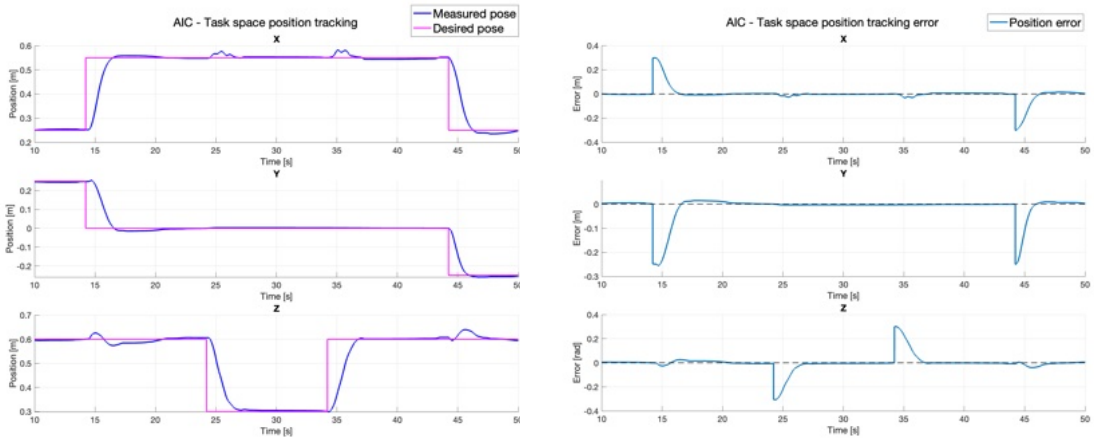
5.4.1.1 Time series plot of the tracking performances of the MRAC, AIC and uAIC controllers



(a) Task-space tracking

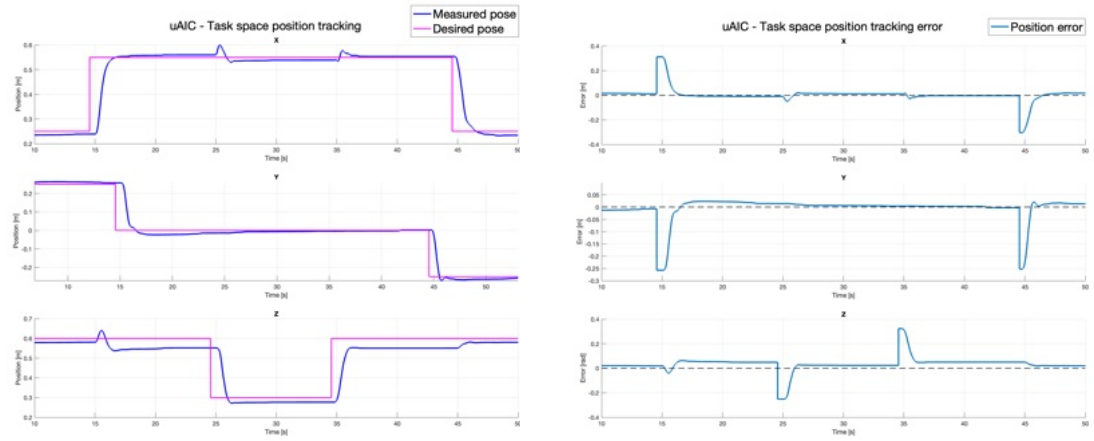
(b) Tracking error

Figure 5.8: Task-space tracking performance of the MRAC. In (a) the actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom). In (b) the task-space tracking error is shown for the three X, Y and Z axes (from top to bottom).



(a) Task-space tracking (b) Tracking error

Figure 5.9: Task-space tracking performance of the AIC. In (a) the actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom). In (b) the task-space tracking error is shown for the three X, Y and Z axes (from top to bottom).



(a) Task-space tracking (b) Tracking error

Figure 5.10: Task-space tracking performance of the uAIC. In (a) the actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom). In (b) the task-space tracking error is shown for the three X, Y and Z axes (from top to bottom).

5.4.1.2 Comparative RMS Performance Analysis of MRAC, AIC, and uAIC Controllers

Tables 5.1, 5.2, and 5.3 respectively show the RMS values of the tracking error in task-space, tracking errors in joint-space, and the torque used over the course of three repeated tests. The grey columns present the results obtained using the MRAC controller, these results are less relevant for the scope of this thesis, as the main comparison is between the AIC and uAIC.

Controller	MRAC	AIC	uAIC	Unit
X	0,0696 (+24%)	0.0563	**0,0490 (-13%)**	m
Y	0,0753 (+54%)	0.0490	**0,0479 (-2%)**	m
Z	0,0809 (+32%)	**0.0612**	0,0655 (+7%)	m
Overall	0,1306 (+35%)	0,0965	**0,0948 (-2%)**	m

Table 5.1: RMS values of tasks-space tracking error for all three axes of the waypoint navigation conducted in task-space. Highlighted in bold is the lowest value among the AIC and uAIC controller. The percentages are referred to the increment/decrement in the RMS when compared to the results obtained by the AIC.

Controller	MRAC	AIC	uAIC	Unit
Joint 1	0,0593 (-13%)	0.0679	***0,0302 (-56%)***	rad
Joint 2	0,1444 (+53%)	0.0944	**0,0660 (-30%)**	rad
Joint 3	0,0875 (+95%)	0.0448	**0,0305 (-32%)**	rad
Joint 4	0,1160 (-12%)	0.1318	**0,0543 (-59%)**	rad
Joint 5	0,0705 (+54%)	0.0458	**0,0209 (-54%)**	rad
Joint 6	0,1120 (-3%)	0.1153	**0,0350 (-70%)**	rad
Joint 7	0,0457 (-65%)	0.1320	**0,0877 (-34%)**	rad
Overall	0,2251 (0%)	0.2564	**0,1360 (-47%)**	rad

Table 5.2: RMS values of the joint-space tracking errors for each one of the joint of the robot. These results are obtained during the waypoint navigation conducted in task-space. Highlighted in bold is the lowest value among the AIC and uAIC controller. The percentages are referred to the increment/decrement in the RMS when compared to the results obtained by the AIC.

Controller	MRAC	AIC	uAIC	Unit
Joint 1	*0,5754 (-4%)*	**0.5985**	*0,9214 (+54%)*	Nm
Joint 2	3,5966 (+10%)	**3.2567**	3,8803 (+19%)	Nm
Joint 3	0,8041 (-18%)	**0.9843**	1,3376 (+36%)	Nm
Joint 4	2,7423 (0%)	**2.7514**	2,9570 (+7%)	Nm
Joint 5	0,9638 (-9%)	1.0577	**0,8773 (-17%)**	Nm
Joint 6	0,5655 (-43%)	0.9891	**0,8954 (-9%)**	Nm
Joint 7	0,3020 (-11%)	**0.3394**	0,3411 (+1%)	Nm
Overall	4,7721 (+2%)	**4,6600**	5,3034 (+14%)	rad

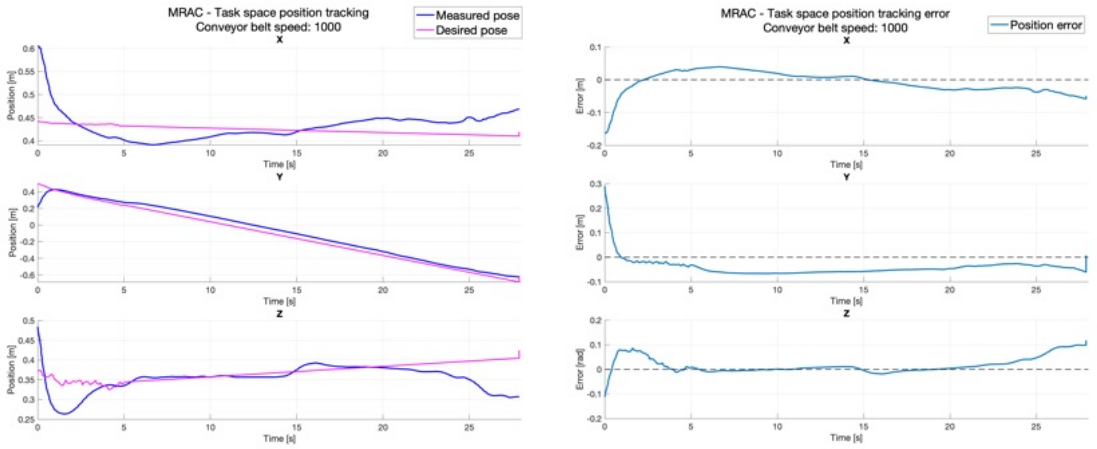
Table 5.3: RMS values of the torques used for each one of the joint of the robot. These results are obtained during the waypoint navigation conducted in task-space. Highlighted in bold is the lowest value among the AIC and uAIC controller. The percentages are referred to the increment/decrement in the RMS when compared to the results obtained by the AIC.

5.4.2 Tracking of a Dynamic Reference

The following results were obtained on the tests described in 4.1.1.2.
For further time series plots of the joint-space tracking performance and the torque usage for all seven joints of the robot can be found in Appendix E.1.2.

5.4.2.1 MRAC dynamic tracking performance

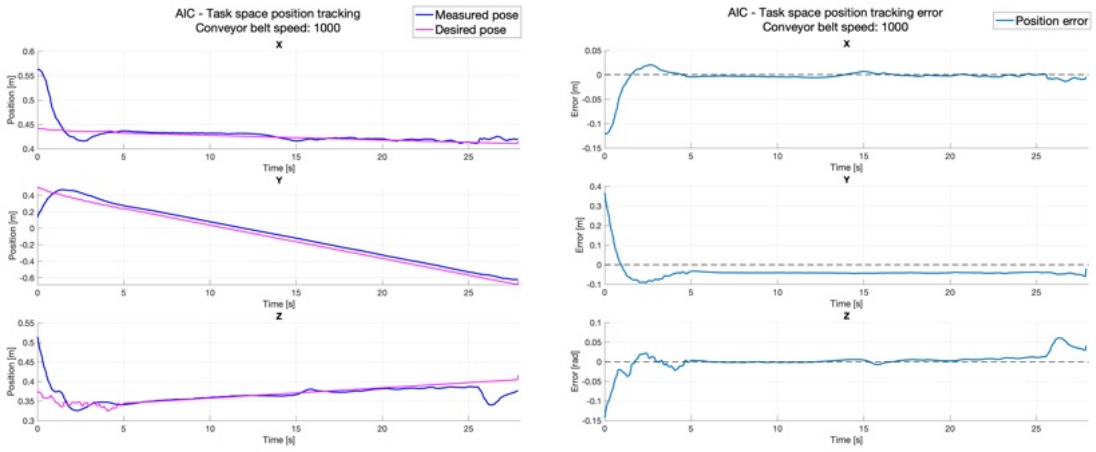
Conveyor Belt Speed: 1000



(a) Task-space tracking

(b) Tracking error

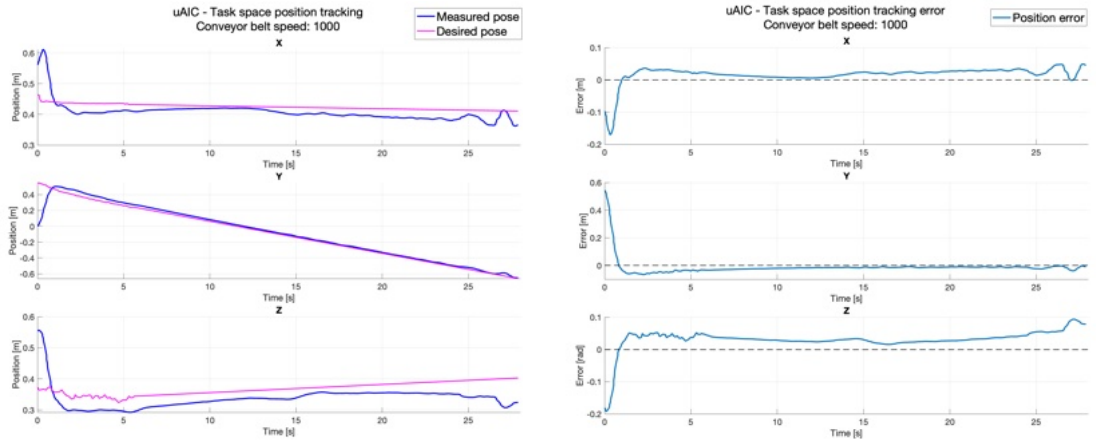
Figure 5.11: Task-space tracking performance of the uAIC when the conveyor belt speed is set to 1000. In (a) the actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom). In (b) the task-space tracking error is shown for the three X, Y and Z axes (from top to bottom).



(a) Task-space tracking

(b) Tracking error

Figure 5.12: Task-space tracking performance of the AIC when the conveyor belt speed is set to 1000. In (a) the actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom). In (b) the task-space tracking error is shown for the three X, Y and Z axes (from top to bottom).

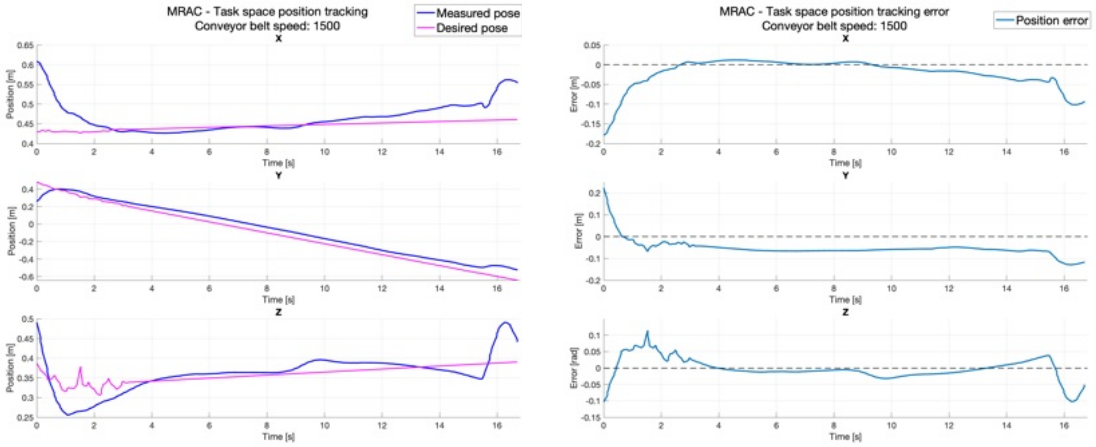


(a) Task-space tracking

(b) Tracking error

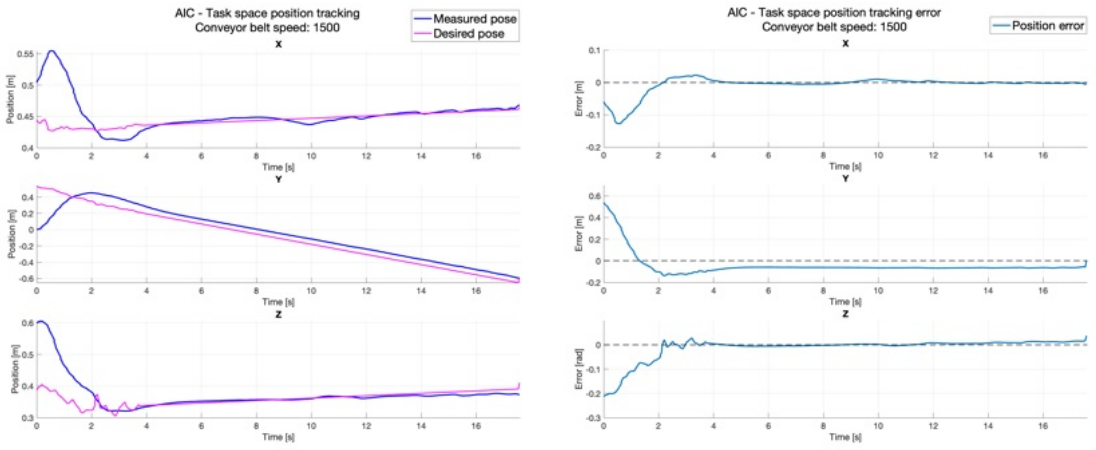
Figure 5.13: Task-space tracking performance of the uAIC when the conveyor belt speed is set to 1000. In (a) the actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom). In (b) the task-space tracking error is shown for the three X, Y and Z axes (from top to bottom).

Conveyor Belt Speed: 1500



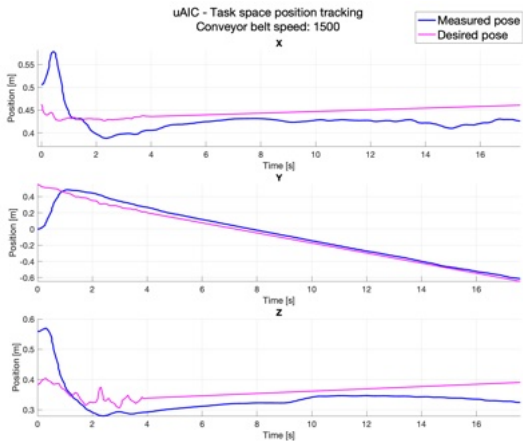
(a) Task-space tracking (b) Tracking error

Figure 5.14: Task-space tracking performance of the uAIC when the conveyor belt speed is set to 1500. In (a) the actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom). In (b) the task-space tracking error is shown for the three X, Y and Z axes (from top to bottom).

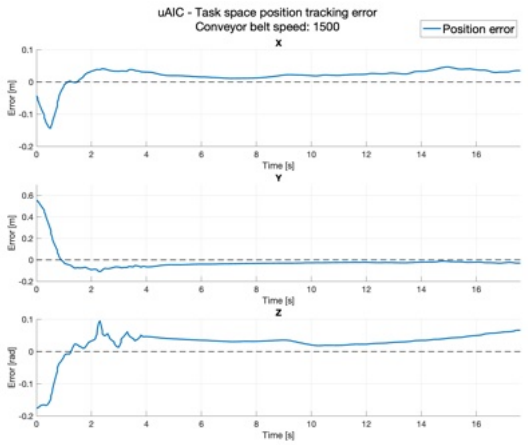


(a) Task-space tracking (b) Tracking error

Figure 5.15: Task-space tracking performance of the AIC when the conveyor belt speed is set to 1500. In (a) the actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom). In (b) the task-space tracking error is shown for the three X, Y and Z axes (from top to bottom).



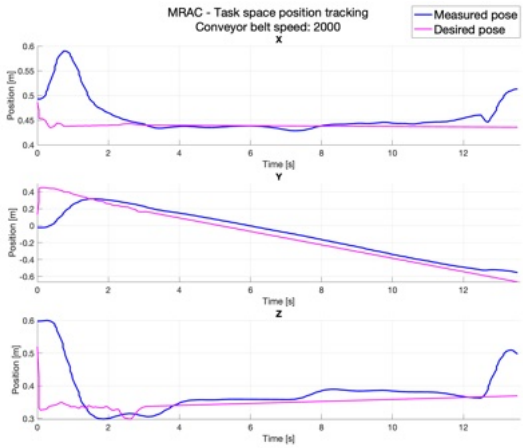
(a) Task-space tracking



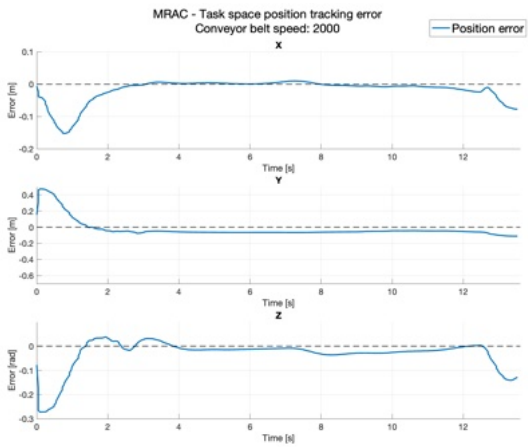
(b) Tracking error

Figure 5.16: Task-space tracking performance of the uAIC when the conveyor belt speed is set to 1500. In (a) the actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom). In (b) the task-space tracking error is shown for the three X, Y and Z axes (from top to bottom).

Conveyor Belt Speed: 2000

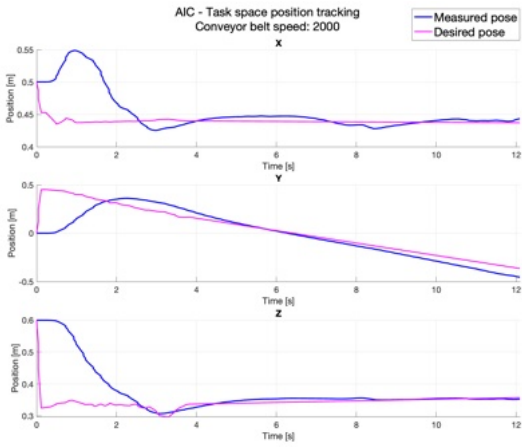


(a) Task-space tracking

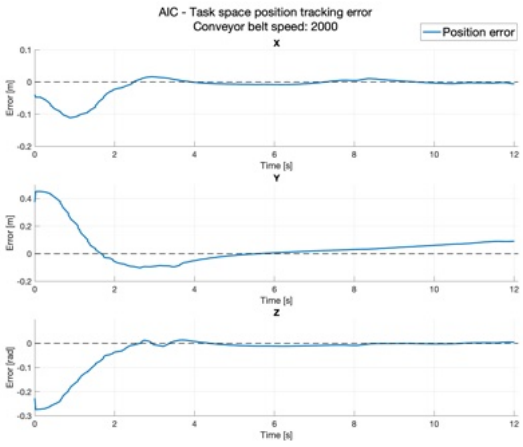


(b) Tracking error

Figure 5.17: Task-space tracking performance of the uAIC when the conveyor belt speed is set to 2000. In (a) the actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom). In (b) the task-space tracking error is shown for the three X, Y and Z axes (from top to bottom).

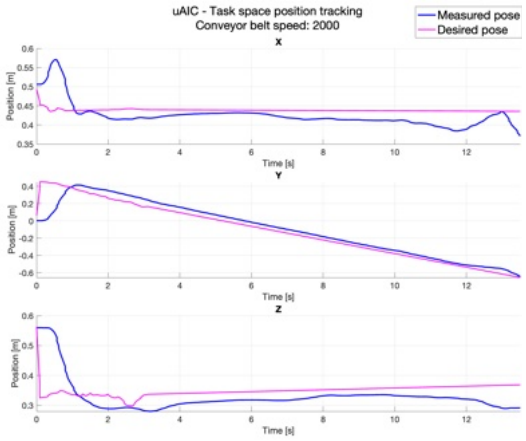


(a) Task-space tracking

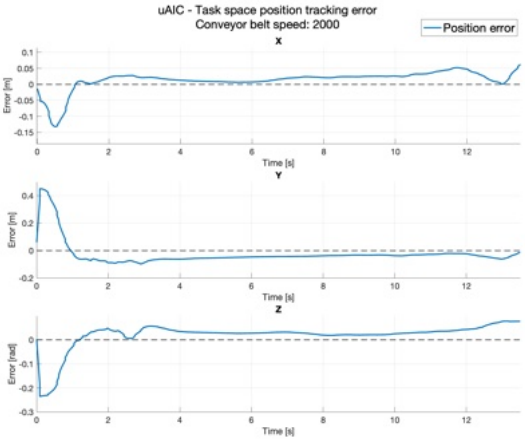


(b) Tracking error

Figure 5.18: Task-space tracking performance of the AIC when the conveyor belt speed is set to 2000. In (a) the actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom). In (b) the task-space tracking error is shown for the three X, Y and Z axes (from top to bottom).



(a) Task-space tracking



(b) Tracking error

Figure 5.19: Task-space tracking performance of the uAIC when the conveyor belt speed is set to 2000. In (a) the actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom). In (b) the task-space tracking error is shown for the three X, Y and Z axes (from top to bottom).

Comparative RMS Performance Analysis of MRAC, AIC, uAIC Controllers

Tables 5.4, 5.5, and 5.6 respectively show the RMS values of the tracking error in task-space, tracking errors in joint-space, and the torque used over the course of three repeated tests. The grey columns present the results obtained using the MRAC controller, these results are less relevant for the scope of this thesis, as the main comparison is between the AIC and uAIC.

Conveyor belt speed	1000			1500			2000			
Controller	MRAC	AIC	uAIC	MRAC	AIC	uAIC	MRAC	AIC	uAIC	Unit
X	0,0340 (+79%)	**0,0190**	0,0301 (+58%)	0,0418 (+46%)	**0,0287**	0,0374 (+30%)	0,0372 (+6%)	0,0351	**0,0340 (-3%)**	m
Y	0,1093 (+87%)	**0,0585**	0,0674 (+15%)	0,1301 (+15%)	**0,1132**	0,1341 (+18%)	0,1495 (+25%)	0,1195	**0,0947 (-21%)**	m
Z	0,0637 (+203%)	**0,0210**	0,0307 (+46%)	0,0675 (+44%)	**0,0470**	0,0717 (+53%)	0,0772 (+7%)	0,0719	**0,0590 (-18%)**	m
Overall	0,0737 (+13%)	**0,0650**	0,0799 (+23%)	0,1529 (+21%)	**0,1259**	0,1566 (+24%)	0,1395 (-8%)	0,1519	**0,1166 (-23%)**	m

Table 5.4: RMS values of the task-space tracking errors for each one of the three axes. These results are obtained during the tracking of the three different trajectories. Highlighted in bold is the lowest value among the AIC and uAIC controller. The percentages are referred to the increment/decrement in the RMS when compared to the results obtained by the AIC.

Conveyor belt speed	1000			1500			2000			
Controller	MRAC	AIC	uAIC	MRAC	AIC	uAIC	MRAC	AIC	uAIC	Unit
Joint 1	0,0749 (-7%)	0,0806	**0,0155 (-81%)**	0,0974 (+29%)	**0,0755**	0,1013 (+34%)	0,0953 (+7%)	0,0892	**0,0206 (-77%)**	Rad
Joint 2	0,0784 (-10%)	0,0871	**0,0303 (-65%)**	0,0994 (-8%)	0,1084	**0,0178 (-84%)**	0,0950 (-10%)	0,1060	**0,0330 (-69%)**	Rad
Joint 3	0,0650 (+15%)	0,0563	**0,0213 (-62%)**	0,0626 (-24%)	0,0827	**0,0361 (-56%)**	0,0776 (+5%)	0,0742	**0,0281 (-62%)**	Rad
Joint 4	0,1652 (-9%)	0,1811	**0,1323 (-27%)**	0,1814 (-11%)	0,2036	**0,0217 (-89%)**	0,0566 (-54%)	0,1220	**0,0191 (-84%)**	Rad
Joint 5	0,0432 (-15%)	0,0507	**0,0323 (-36%)**	0,0730 (+8%)	**0,0679**	0,1638 (+141%)	0,0729 (+23%)	0,0593	**0,0396 (-33%)**	Rad
Joint 6	0,1695 (+4%)	0,1634	**0,1309 (-20%)**	0,1436 (-14%)	0,1675	**0,0314 (-81%)**	0,0502 (-47%)	0,0951	**0,0575 (-40%)**	Rad
Joint 7	0,0547 (-69%)	0,1780	**0,0930 (-48%)**	0,0589 (-71%)	0,2027	**0,1519 (-25%)**	0,0731 (-65%)	0,2087	**0,1182 (-43%)**	Rad
Overall	0,2772 (-17%)	0,3332	**0,1302 (-61%)**	0,2926 (-22%)	0,3735	**0,1642 (-56%)**	0,2013 (-35%)	0,3094	**0,1858 (-40%)**	Rad

Table 5.5: RMS values of the joint-space tracking errors for each one of the joint of the robot. These results are obtained during the tracking of the three different trajectories. Highlighted in bold is the lowest value among the AIC and uAIC controller. The percentages are referred to the increment/decrement in the RMS when compared to the results obtained by the AIC.

As observed in the time series plots in Chapter 5.4.2, and the RMS table 5.4, the AIC performed better in task-space in the tracking challenge at conveyor belt speeds of 1000 and 1500, whereas it performed worse at a speed of 2000, when compared to the uAIC. However, the results in Table 5.5 show that uAIC consistently reduced joint-space tracking errors across nearly all joints and tracking challenges. It reported an overall reduction of the joint-space tracking error of $61\%, 56\%$ and 40% for the tracking challenge at speed 1000, 1500 and 2000 of the conveyor belt, respectively, compared to the AIC. One would expect that a reduction in joint-space errors would lead to a reduction in task-space errors. The reasons why this did not happen will be discussed in Chapter 6.3.1.2.

Conveyor belt speed	1000			1500			2000			
Controller	MRAC	AIC	uAIC	MRAC	AIC	uAIC	MRAC	AIC	uAIC	Unit
Joint 1	0,8741 (+8%)	**0,8102**	1,4069 (+74%)	1,1706 (+36%)	**0,8592**	1,9189 (+123%)	1,2542 (+52%)	**0,8277**	1,8048 (+118%)	Nm
Joint 2	3,7424 (+1%)	3,7216	**3,7181 (0%)**	3,6890 (+3%)	**3,5931**	*3,9247 (+9%)*	3,7109 (+8%)	**3,4211**	3,5125 (+3%)	Nm
Joint 3	0,8000 (-3%)	**0,8212**	1,1027 (+34%)	0,9030 (-9%)	**0,9887**	*1,4314 (+45%)*	1,0976 (+26%)	**0,8741**	1,6849 (+93%)	Nm
Joint 4	2,3751 (-7%)	**2,5449**	2,6597 (+5%)	2,9470 (0%)	2,9486	**2,9304 (-1%)**	2,8861 (-1%)	2,9155	**2,7067 (-7%)**	Nm
Joint 5	0,8754 (+3%)	**0,8540**	1,0994 (+29%)	0,8801 (-4%)	0,9137	**0,8447 (-8%)**	0,9246 (+14%)	**0,8079**	0,8826 (+9%)	Nm
Joint 6	1,0661 (+19%)	0,8958	**0,7612 (-15%)**	0,7998 (-9%)	0,8760	**0,8130 (-7%)**	0,7274 (-17%)	0,8814	**0,7685 (-13%)**	Nm
Joint 7	0,4872 (+8%)	**0,4520**	0,4810 (+6%)	0,4739 (+6%)	**0,4490**	0,5077 (+13%)	0,5175 (+18%)	**0,4382**	0,5172 (+18%)	Nm
Overall	4,8157 (-0,4%)	**4,8467**	5,1101 (+6%)	5,1106 (+2%)	**5,0124**	5,5995 (+12%)	5,1507 (+7%)	**4,8244**	5,2342 (+8%)	Nm

Table 5.6: RMS values of the torques used for each one of the joint of the robot. These results are obtained during the tracking of the three different trajectories. Highlighted in bold is the lowest value among the AIC and uAIC controller. The percentages are referred to the increment/decrement in the RMS when compared to the results obtained by the AIC.

5.5 Comparison of the Performances on a Simulated E-Grocery Context

In this section the results obtained with the experiments described in 4.1.2.1 and 4.1.2.2 are presented.
As mentioned in Section 1.3, the performance metrics chosen to evaluate the performance of the three controllers in the e-grocery context are:

- Success rate;
- Time needed to complete a task.

A task is considered successfully completed when the object is correctly grasped and placed within the desired time frame.

For grasping static objects, the maximum allowed time is 60 seconds. In the case of the dynamic e-grocery task, where the object is conveyed at different speeds, there is no maximum allowed time, as this is naturally limited by the speed of the object and the dimensions of the grasping area, as discussed in Section 4.1.2.2.

5.5.1 Grasping of Static Objects From Unknown Locations

Following, the summary of the results obtained in the e-grocery task described in Chapter 4.1.2.1.
These results were obtained by running 15 tests for each area and each controller, resulting in 45 tests per controller and a total of 135 tests. In this section only the graphs that highlight the overall results obtained with the experiments will be shown, for in depth details about experiments' data the reader is referred to Appendix E.2.1.
Below Table 5.7 summarizes the errors encountered while conducting this set of experiments.

Error	Error Description	Error Reason
Too Slow	The overall time needed to complete the pick and place operation exceeded the maximum allowed time of 60 seconds.	The controller needed a lot of time to converge to the desired set-points.
Failed to Converge to Set-Point	The controller failed to bring the end effector to the desired waypoint (e.g., object location or placing position). This resulted in a failure to grasp the object or a failure to place it correctly, such as by releasing it from a higher position than intended.	Inaccuracy of the controller. For example, the controller overshot the desired position, causing the end-effector to move to unintended locations, but still passing through the desired picking position within the set thresholds. To better understand the logic behind this task, the reader is referred to Figure C.1
Cartesian Reflex	The end-effector either approached the object or placing location with a high velocity or continued pushing down towards the ground with an increasing force.	The high force applied by the gripper towards the ground resulted in a significant external force. When this external force exceeded a certain threshold, the FCI automatically blocked the robot, making it impossible to continue the task.

Table 5.7: Errors encountered while grasping static objects from unknown locations.

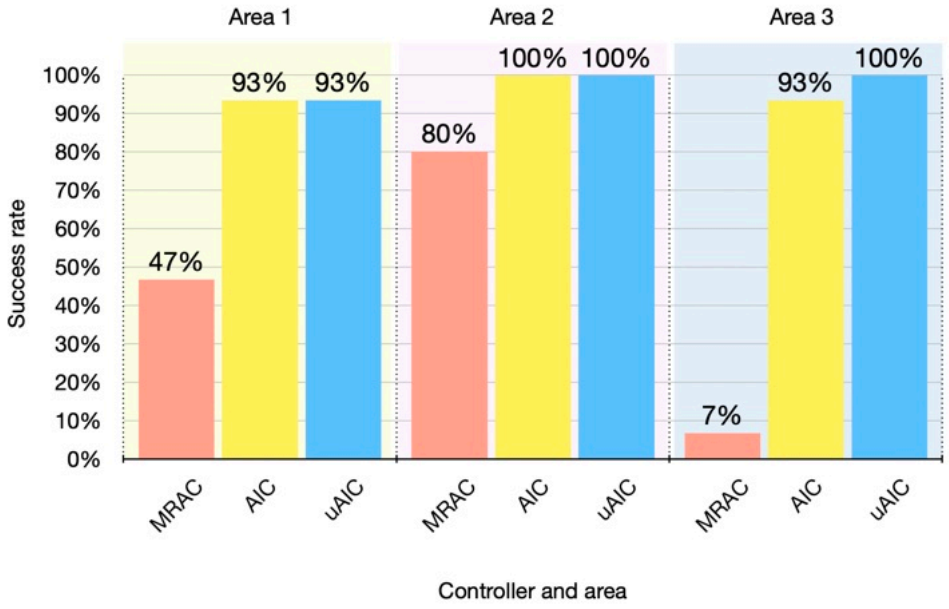


Figure 5.20: Success rate per controller, per working area, for grasping static objects from unknown locations.

In Figure 5.20, the column plot shows the number of successes per controller per area. A task is considered successful if the object is successfully grasped, placed correctly at the designated point, and completed within 60 seconds.

The success rates demonstrates that both AIC and uAIC consistently achieve high performance across all areas. In Area 1, both controllers reach a success rate of 93%. This trend continues in Area 2, where both AIC and uAIC achieve a perfect success rate of 100%. The consistency of these controllers is further highlighted in Area 3, where AIC maintains a high success rate of 93% and uAIC reaches 100%. These results indicate

that both AIC and uAIC are highly effective, with uAIC being more consistent across the entire working area. Conversely, the MRAC performs poorly across all areas except Area 2, achieving success rates of 40%, 80%, and 7% in Areas 1, 2, and 3, respectively.

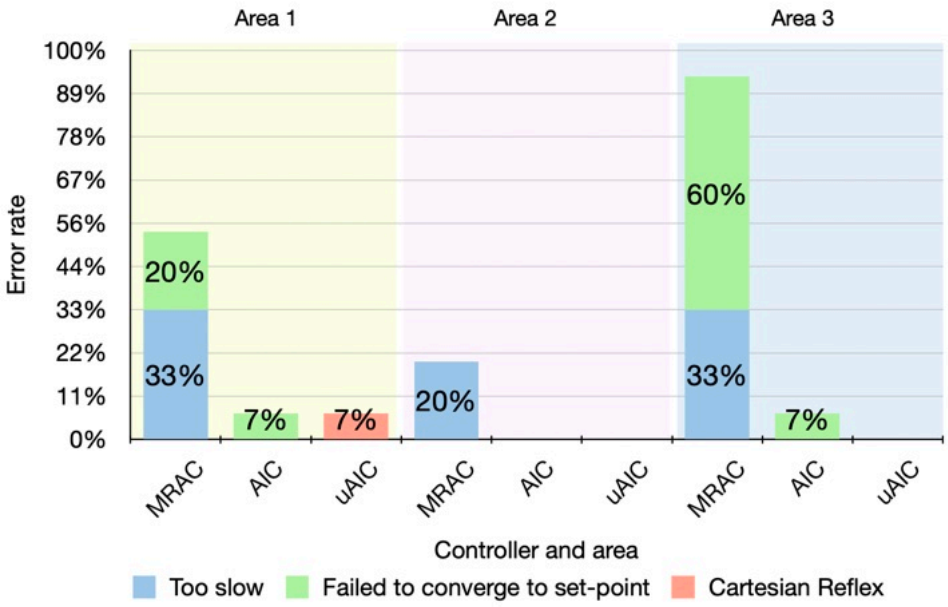


Figure 5.21: Error rate per controller, per working area, for grasping static objects from unknown locations.

The error rates shown in Figure 5.21 provide further clarity on the reliability of AIC and uAIC. In Areas 1 and 3, the AIC failed to converge to the desired position in 7% of the experiments. The uAIC, on the other hand, only failed in Area 1 in 7% of the experiments 1 due to ”Cartesian Reflex”.

Lastly, the MRAC has demonstrated to be the least successful, especially in side areas 1 and 3, where 20% and 60% of the tasks failed due to the controller not being able to reach the desired object position (”Failed to Converge to Set-Point”). The lack of precision has also been the cause of the excessive time needed to reach the desired pose and consequently complete the pick-and-place operation.

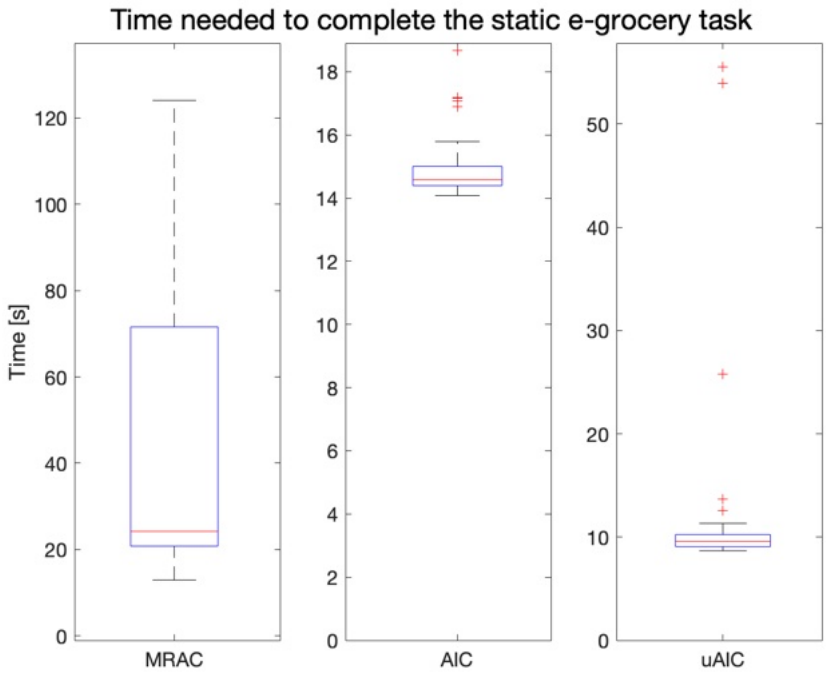


Figure 5.22: Time needed to detect, pick and place a static object from an unknown location

5.5.2 Grasping of Moving Objects From Unknown Locations

Below is a summary of the results obtained in the e-grocery task described in Chapter 4.1.2.2. These results were obtained by running 30 tests for each speed and each controller, resulting in 90 tests per controller and a total of 270 tests.
In this section only the graphs that highlight the overall results obtained with the experiments will be shown, for in depth details about experiments' data the reader is referred to Appendix E.2.2.
While running the tests, various types of errors were observed. These errors are reported and described in Table 5.8.

Error	Error Description	Error Reason
Cartesian Reflex	The end-effector either approached the object or the placing location with a high speed or continued pushing down towards the ground with an increasing force. These errors do not include Cartesian Reflex errors caused by a wrong object pose prediction.	The inaccuracy of the controller made it so that the end effector applied a high force down towards the ground, resulting in a significant external force applied from the object/table to the gripper. When this external force exceeded a certain threshold, the FCI automatically blocked the robot, making it impossible to continue the task.
Prediction Error	The pose of the object published by the Kalman predictor is inaccurate. A high error on the Z coordinate caused the robot to either miss the object or push on it so much to cause a Cartesian Reflex error, however since the Cartesian Reflex was caused by a wrong predicted pose they have been classified as prediction errors.	The Kalman predictor bases its estimate of the object pose on data recorded by the camera in a limited area of the conveyor belt. The object pose recorded by the camera is also noisy. If either the X or Z coordinate shows an increasing or decreasing trend toward the end of the detection area, due to the inherently noisy nature of the signal, the Kalman filter assumes these coordinates are linearly growing over time, thus providing an incorrect prediction of the object pose.
Failed to Converge to Set-Point	The controller failed to bring the end effector to the desired waypoint (e.g., object location or placing position). This resulted in a failure to grasp the object or a failure to place it correctly, such as by releasing it from a higher position than intended.	Inaccuracy of the controller. Either, the controller overshot the desired position, causing the end-effector to move to unintended locations, but still passing through the desired picking position within the set thresholds. Or the error, on one or more of the three axis, was too big and the end-effector could not reach the desired pose. To better understand the logic behind this task, the reader is referred to Figure C.2

Table 5.8: Errors encountered while grasping moving objects from unknown locations.

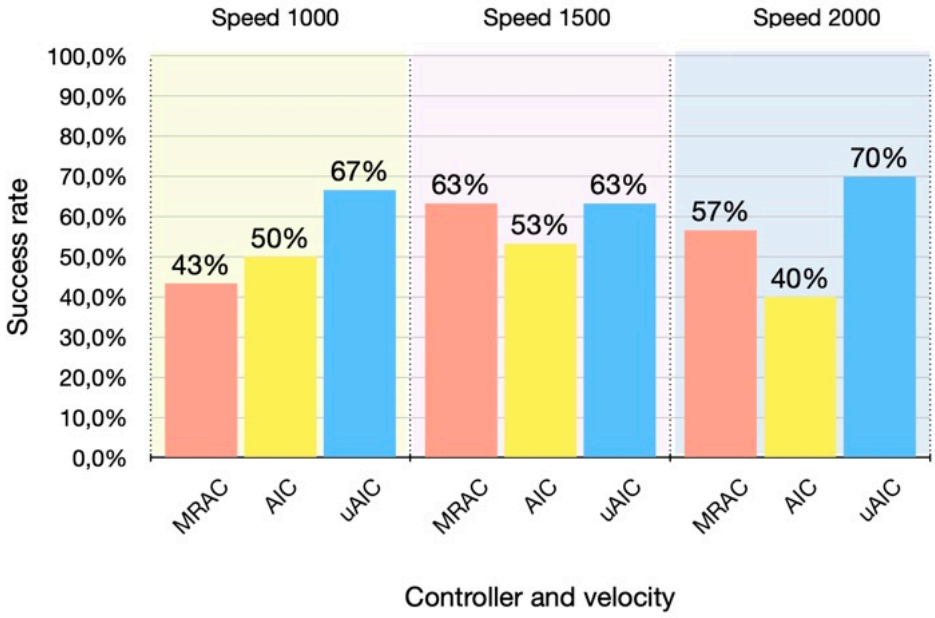


Figure 5.23: Success rate per controller, per different conveyor speed, for grasping moving objects from unknown locations.

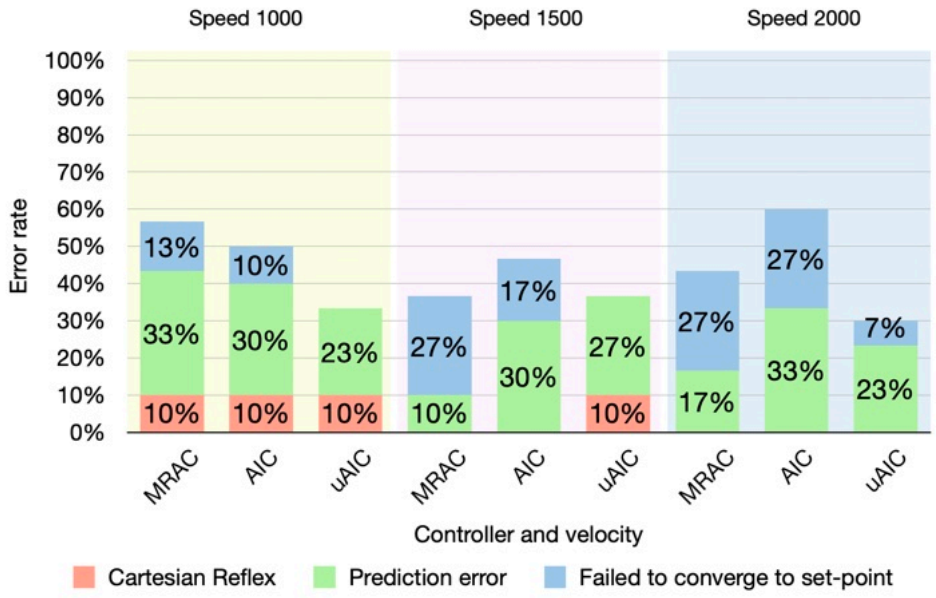


Figure 5.24: Error rate per controller, per different conveyor speed, for grasping moving objects from unknown locations.

Figure 5.24 shows the rate of each registered error against the total number of tasks in each set of experiments. Each column represents the overall error rate for each controller and conveyor belt speed, over the total number of experiments.

For example, in the first column, Cartesian Reflex: 10%, indicates that the MRAC failed 3 out of 30 experiments due to the Cartesian Reflex error. The sum of all errors is 56%, while the remaining 44% represents the successfully completed tasks.

For some of the columns, the rates might not add up to 100% if the data from Figure 5.23 is also taken into consideration. This discrepancy is due to rounding applied to the data to enhance the readability of the plots.

5.5.2.1 Success rate excluding prediction errors

As shown in Figure 5.24, approximately one-third of the errors encountered in nearly all experiments result from incorrect object pose predictions by the Kalman Filter.

Figure 5.12 illustrates the impact of these incorrect predictions, especially along the Z-axis. The Kalman Filter inaccurately predicts a continuous increase in the grasping height of the object, which is implausible given the nature of the object and its environment. Since these errors cannot be attributed to the controllers and predicting the Z-axis pose is unnecessary in this scenario, the experiments were repeated with the Kalman Filter restricted to predicting only the X and Y coordinates of the object's pose. In this case, the grasping height was recorded and saved as the object passed through the detection area and re-used later when the object entered the grasping area.

The results of these new experiments are shown in Figure 5.25 and Figure 5.26. A total of 30 tests were conducted for each speed of the conveyor belt and controller, amounting to 90 tests per controller and a total of 270 tests.

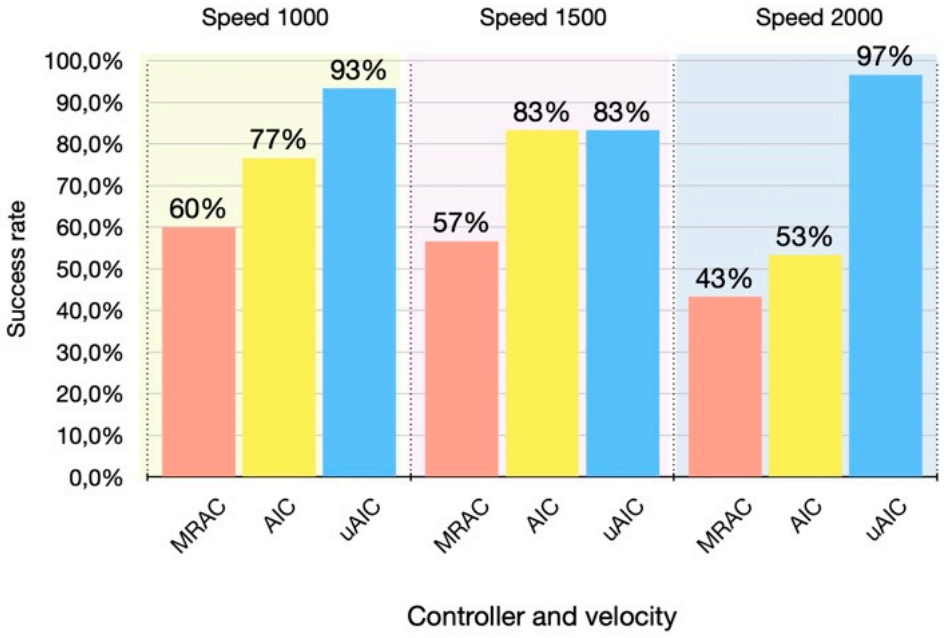


Figure 5.25: Error rate per controller, per different conveyor speed, for grasping moving objects from unknown locations.

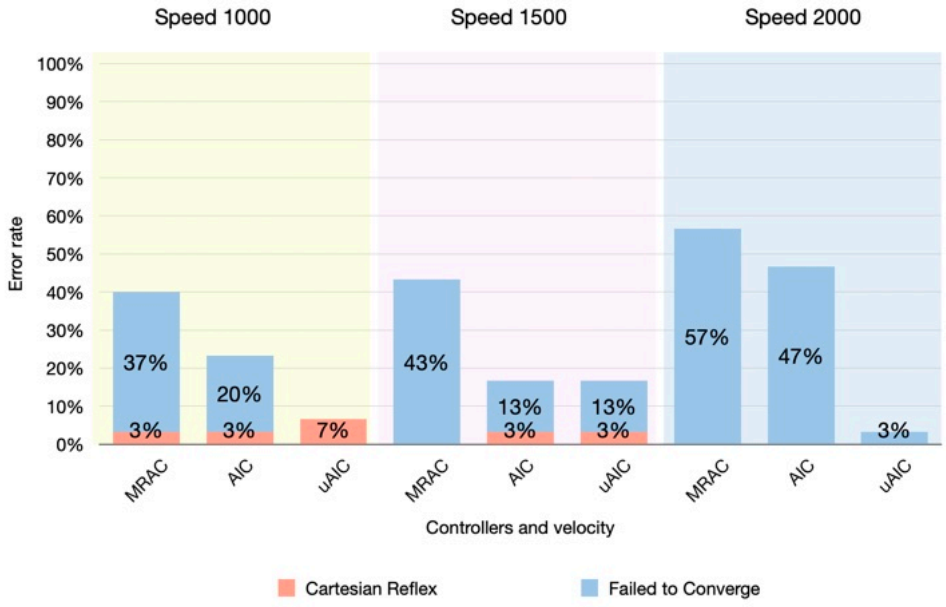


Figure 5.26: Error rate per controller, per different conveyor speed, for grasping moving objects from unknown locations.

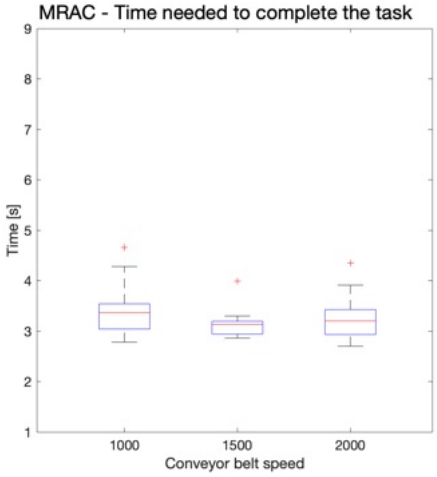
Upon filtering out prediction errors, the success rates improved significantly. The AIC achieved a success rate of 77% and 83% in the first two challenges. However, as the object's speed increased, the controller's performance deteriorated, with a success rate

of only 53% at conveyor speed 2000.

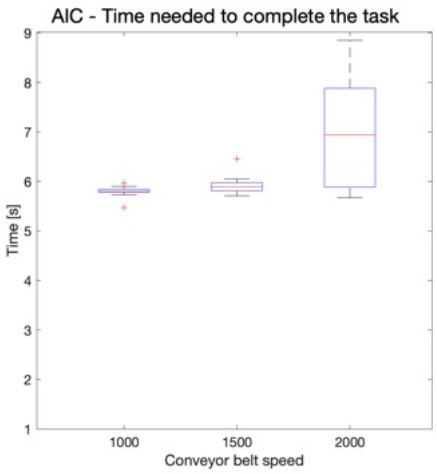
The MRAC exhibited a decreasing trend in success rates as the conveyor belt speed increased.

In contrast, the uAIC consistently outperformed all the other controllers in all three challenges, unaffected by the increase in the object's speed.

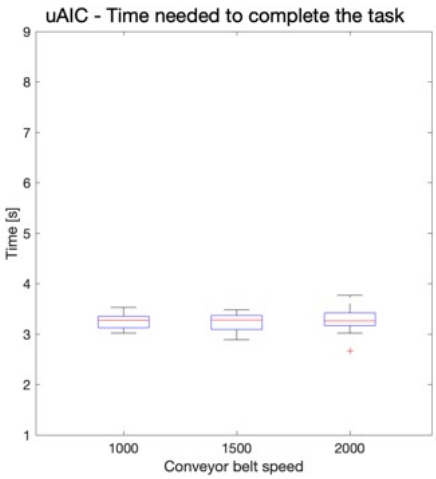
The error rates in Figure 5.26 reveals that the uAIC controller had the lowest overall error rates across all tested speeds. For all three controllers, the rate of abrupt blocks due to "Cartesian Reflex" error remained coherent with what observed in the previous set of experiments (Figure 5.24), whereas the rate of the "Failed to Converge to Set-Point" has seen a radical increase, especially for the MRAC and the AIC.



(a) MRAC



(b) AIC



(c) uAIC

Figure 5.27: Time needed to pick and place a moving object from random locations at various speed.

Figure 5.27 time required to complete pick-and-place tasks also varied among the con-

trollers.
Overall, the MRAC and uAIC demonstrated to be the fastest and most consistent controllers, with a median completion time oscillating between 3.13 seconds and 3.65 seconds for the MRAC and between 3.26 seconds and 3.28 seconds for the uAIC. The AIC was the slowest, with a median time of 5.8 seconds for the first two challenges, and a median time of 6.94 seconds for the last challenge, exhibiting high variability.

6 Discussion

In this chapter, the results obtained from the experiments and the implementation choices of some of the components of the system will be discussed.

6.1 Filtering of the IK Reference

To allow the system to operate with task-space references the already implemented inverse kinematics resolver, Trac-IK, was utilized.
This solver, although it has the advantage that it checks for joint limits violation and avoids self-collision, it had the drawback that sometimes provided different joint configurations for the same task-space position, as discussed in Chapter 4.2.2 and seen in Figure 5.1. This problem particularly occurs when the target pose is near the limits of the reach zone of the robot.
The resulting significant changes in joint configurations causes spikes, high frequency noise, in the low-pass position reference (Figure 5.1). This did not cause any measurable problems when conducting experiments with the AIC. However, the uAIC has demonstrated to be much more susceptible to this noise, resulting in the safety block of the robotic arm as shown in Figure 4.8. To address this, other IK solvers were considered. However, due to the extensive time required to research, study, and implement a new IK solver, the solver from [19] was initially implemented. Unfortunately, this solver did not account for self-collision when evaluating joint configurations given any task-space pose, often resulting in self-collision of the robotic arm.

Therefore, implementing a filter for the IK reference was deemed the faster and easier solution. A first-order low-pass filter was implemented to smooth these spikes in position references. The filter parameters, described in Chapter 4.2.2.1, were chosen to ensure the filtered signal remained true to the original reference, effectively rejecting high-frequency spikes without excessively damping the signal.
The application of the low-pass filter on the real robot confirmed, as per the Matlab findings, that it successfully attenuated most of the high-frequency noise. However, abrupt motions of the robot's joints were observed during the start-up phase of the controller when using the low-pass filter. These abrupt motions were not present when the controller operated without the low-pass filter. This issue is likely due to a problem in the implementation of the filter and how it generated the initial joint references for the controller. Positioning the robot in the default position (Figure F.3) mitigated this problem.

To ensure a fair comparison among the three controllers, the filter was applied to the reference passed to each controller. Although, as mentioned, the AIC did not inherently require a filter, applying it was necessary for consistency in the results obtained in the experiments.

6.2 Tuning of the uAIC

As discussed in Section 4.2.2.2, the initial implementation of the uAIC utilized the tuning parameters provided by [4].
To mitigate potential damage, the end-effector was removed from the robot throughout the implementation phase. Once the implementation was deemed successful, the gripper and its support were reattached to the robot for testing the controllers in real-life scenarios. It

was observed that, when controlled by the uAIC, the robot's behavior differed when operating with the end-effector attached compared to when it was detached. The addition of the gripper altered the kinematic chain of the robot and consequently affected its dynamics. The impact of this additional mass was particularly noticeable in the joints with the axes parallel to the ground, specifically the second, fourth, and sixth joints (Figure 5.3).

The resultant effect was that the end-effector struggled to maintain the desired commanded pose and orientation, leading to potential collisions with the table or conveyor belt. This result was unexpected since the uAIC, by not needing any description of the system, should theoretically be unaffected by changes in the inertia of the robotic arm, as also reported by [5]. The reason for this discrepancy was attributed to either a probable different tuning of the controllers in both [4] and [5], or a possible mistake in the implementation of the controller in ROS, this problem still requires further investigation. Due to time constraints, it was decided to re-tune the controller to achieve a satisfactory response and proceed with the experiments.

Initially, an attempt was made to complete the Matlab model of the controller to enable open and closed-loop simulations applied to a model of the robot. However, this approach was abandoned early in the process due to the extensive time required for the implementation. Given the project's time constraints, resources were reallocated to other tasks. Consequently, a trial-and-error approach for tuning was adopted, which was deemed to provide satisfactory results while reducing the time needed for controller tuning. The uAIC tuning parameters were adjusted to those specified in Chapter 4.2.2.2, Table 4.2.

The effects of this new tuning of the uAIC can be seen in Chapter 5.2. It can be seen how with the original tuning (Figure 5.2), the controller followed the joint-references closely but with a certain level of damping, but overall converging to the desired joint angle. When the gripper was attached (Figure 5.3) the joints which had the axis parallel to the ground (Joint 2, 4 and 6) were the most affected by the added weight.
After the controller was tuned the response became much more aggressive (Figure 5.4), meaning it followed the reference much more accurately, with very little damping and an error close to zero when the gripper was detached. In the same way as per the original tuning a slightly higher error on Joint 2, 4 and 6 was observer when the gripper was attached (Figure 5.5). This error could not be reduced, and increasing the k_i tuning parameter in an attempt to reduce it, suddenly lead to a violation over the maximum speed allowed.
It should be noted, however, that while this approach has led to some improvements in the overall tracking performance of the controller, it also has certain drawbacks.

The first drawback is that in rare instances, especially for poses that fall on the back of the robot base or on the far side of it, the robot would sometimes violate the joint speed limits while reaching those poses.

The second drawback, which is also an advantage, is the more aggressive response that leads to faster movements of the robot. On one hand, this enables the robot to complete tasks in less time. On the other hand, it could potentially be problematic when handling delicate food products, especially if they are not grasped properly.

6.3 Performance Evaluation Between MRAC, AIC and uAIC

6.3.1 Tracking Performances in Task-Space and Joint-Space

In this section the results showed in 5.4 are discussed.

6.3.1.1 Waypoints Navigation

The results obtained in navigating through static waypoints indicate that, overall, both the AIC and uAIC outperformed the MRAC, although the uAIC performed slightly better than the AIC. This can be observed in the time series plots 5.9 and 5.10, as well as in the RMS tables Tables 5.1 and 5.2.

6.3.1.2 Tracking of a Dynamic Reference

The observed counterintuitive results seen in 5.4.2, where improvements in joint-space tracking do not correspond to improvements in task-space tracking, can be explained by considering the interplay between the IK resolver and the controller dynamics.

As detailed in Chapter 5.1 and discussed in 6.1, the IK resolver often generates joint-space references containing high-frequency noise, which induces noticeable vibrations in some of the robot's joints. While applying a filter to these references reduces much of the high-frequency noise, small spikes still remain.

The uAIC controller, characterized by a rapid response to change in the reference, closely follows these joint-space references, including the residual spikes. This responsiveness causes small, rapid adjustments in the joint configurations, leading to significant deviations of the end-effector from the desired pose in task-space due to the robot's kinematic sensitivities.
This phenomenon is also observed and discussed in [21] where the authors have observed how errors in the joint-space propagate to the task-space, impacting the accuracy of the end-effector's trajectory.
Thus, despite better joint-space tracking, the uAIC's task-space performance suffers due to the oscillatory behaviour of the IK reference.

Conversely, the AIC controller, with its more damped response, inherently smooths out the oscillations and spikes present in the IK references. This damping effect prevents rapid adjustments in the joint configurations, keeping the end-effector closer to the desired task-space trajectory and resulting in better task-space tracking performance.
It is worth noting that while the RMS values of the tracking error along the y-axis favor the AIC at conveyor speeds of 1000 and 1500, the error remains consistent throughout the entire test and increases with the conveyor belt speed, ad it can be seen in Figure E.11 and Figure E.20.
In contrast, although the uAIC may exhibit higher oscillations during the transient phase, leading to a higher RMS of the tracking errors, it tends to follow the reference trajectory, along y, more closely and reduces the error more effectively than the AIC.

6.3.2 Performances Evaluation on an E-Grocery Simulated Context

In this section, the results presented in 5.5 are discussed.

As shown in the results section 5.5.2, the uAIC has proven to be the most consistent and reliable controller in both e-grocery challenges, providing an overall lower completion

time and a higher success rate compared to the AIC.

Seen that the performances, in terms of success rate, of the AIC and uAIC are comparable in tasks where the objects to be grasped are static, one might consider trading the slightly lower precision and longer completion time of the AIC for its reduced torque usage. However, this trade-off does not hold when grasping moving objects.

The uAIC has consistently reported higher success rates across tests conducted at different speeds and has shown a reduction of approximately 50% in completion time. Its success is mainly due to its faster tracking speed, as reported in [4], and the re-tuning of the controller, brought out during this project, which has made the controller more responsive and consequently have made the robot's movement faster.
As shown in Figure C.2, the gripper can grasp the object only if the controller brings the end-effector close to the target position within certain thresholds (see Table C.2). Despite some oscillatory behavior, the uAIC managed to reduce the tracking error in most tests. In contrast, the AIC failed some tasks because it could not effectively reduce the error along the y-axis, as explained in the previous section (6.3.1.2). This resulted in the ”Fail to converge” error reported in Figure 5.26.

6.4 Limitations and Future Work

In this section, the limitations of the system that emerged during the discussion of the results are highlighted. These limitations will also serve as a valuable starting point for discussing potential future work.

Variability in IK Solutions
The Trac-IK solver introduced variability in joint configurations for the same task-space positions, especially near the robot's reach limits. This variability resulted in high-frequency noise in the low-pass references, affecting the performance of the uAIC controller more than the others. Although implementing a low-pass filter adds complexity and potential computational cost to the system, it has proven effective in reducing high-frequency noise. However, some variability in the provided IK solution remains.

Future work could explore various directions. For instance, researching and implementing better methods of filtering the IK reference could be beneficial. Additionally, while Trac-IK was chosen for its advantages, such as checking for joint violations, self-collisions, and fast evaluation speed, it may be worthwhile to study and implement alternative approaches to IK reference evaluation that better suit the requirements posed by the controllers when tracking dynamic references.

Limits of the uAIC
The uAIC controller, despite its superior performance, showed sensitivity to dynamic changes in the system, such as the addition of the end-effector. This change altered the kinematic chain and affected the robot's dynamics, necessitating re-tuning of the controller. This sensitivity limits the plug-and-play capability of the controller and requires additional calibration when significant modifications are introduced to the kinematic chain. Discrepancies between the expected and actual performance were observed, likely due to potential errors in the implementation process or different tuning of the controller (when compared to [4] and [5]).

To this regards, a possible future work could see the implementation of an online optimization methods of the controller's parameters to avoid a manual re-tuning of the controller anytime the dynamic of the robot changes. Another possible improvement could be done in smoothing the control action by implementing a smoothing prior of the control action as suggested by the authors in [4].

Although the uAIC showcased better tracking performance in joint-space, especially with dynamic references, these improvements came at the cost of increased torque usage.
A possible future improvement in this aspect could be the implementation of a control cost, similar to an Linear Quadratic Gaussian (LQR), as suggested by the authors of [4].

Limited Scope of Testing
The experiments were conducted under specific conditions and may not fully represent all possible real-world scenarios. The testing focused on particular object speeds and types of dynamic changes, limiting the generalizability of the findings to other scenarios or more diverse operational conditions.

Environment
Some aspects of the environment are fixed and cannot be altered. The hardware and software must remain unchanged because the system is specifically designed for them. Furthermore, this project relies on a fixed position for each component, such as the camera, conveyor belt, and robot arm. Changing their relative positions could cause the object detector to analyze incorrect regions and potentially send wrong object coordinates to the supervisor.

7 Conclusions

In this project, a model-free adaptive controller based on the AIF framework, specifically the uAIC, was implemented on the Franka Emika Panda robot manipulator. The goal was to compare its performance against its predecessor, the AIC, and a model-based adaptive controller, the MRAC, in the context of e-grocery.

Instead of passing position references in joint-space to the controllers, as done in previous works [4] and [5], this project used position references in task-space. To the best of our knowledge, this is the first time the uAIC has been tested in task-space. A similar study was conducted by [6] for the MRAC and AIC. Because the controllers only accepted joint-space references, an IK solver, Trac-IK, was used to convert task-space references into joint-space references suitable for the controllers.

To assess performance, two sets of tasks were designed. The first set aimed to analyze the tracking performance of the controllers in both joint-space and task-space, using both constant and time-varying task-space references. The second set evaluated the performance of the controllers in a simulated e-grocery environment with both static and moving objects. For static objects, coordinates were passed directly from the camera, while for moving objects, coordinates were provided by a Kalman Filter implemented in [6].

Overall, the results of the benchmark tests showed that the AIC outperformed both the MRAC and uAIC in task-space tracking of a static reference while also using less torque. When tracking time-varying references, the AIC still outperformed the other controllers at lower speeds. However, as the speed increased, the uAIC exhibited a lower tracking error compared to the other controllers.
Interestingly, these results were obtained despite the uAIC consistently reducing tracking error in joint-space in all tests. The cause was attributed to noise in the Trac-IK generated joint-space references. The uAIC, with its fast response, tended to track the residual spikes in these references, leading to deviations of the end-effector pose from the desired task-space position.
For the e-grocery tasks, both the AIC and uAIC outperformed the MRAC, with the uAIC demonstrating higher reliability and consistency, despite its observed poor performance in trajectory tracking in task-space.
In the grasping of static objects, the AIC and uAIC had similar performance, in term of success rate, with the uAIC requiring on average 30% less time to complete a task.
However, when grasping moving objects, the uAIC proved to be the most reliable, consistent, and faster controller. This can be attributed to the uAIC's faster response [4], which, despite some oscillations, helped reduce the tracking error along the y-axis more effectively than the AIC, subsequently increasing the chances of reaching the desired moving target. The uAIC demonstrated a high success rate across all challenges and speeds, whereas the AIC success rate decreased as the speed of the conveyor belt increased.

This research indicates that the uAIC clearly has some advantages over the AIC when deployed in an e-grocery context. However, being this the first implementation of the controller in such a context, further work (6.4) is still required.

Bibliography

[1] R. Syed et al. “Robotic Process Automation: Contemporary themes and challenges”. In: *Comput. Ind.* 115 (2020), p. 103162. DOI: 10.1016/j.compind.2019.103162.

[2] J. Davidson et al. “Robotic Manipulation for Specialty Crop Harvesting: A Review of Manipulator and End-Effector Technologies”. In: 2 (2020), pp. 25–41. DOI: 10.35251/gjaas.2020.004.

[3] Corrado Pezzato, Riccardo Ferrari, and Carlos Hernández Corbato. “A Novel Adaptive Controller for Robot Manipulators Based on Active Inference”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2973–2980. DOI: 10.1109/LRA.2020.2974451.

[4] Mohamed Baioumy et al. *Unbiased Active Inference for Classical Control*. 2022. arXiv: 2207.13409 [cs.RO].

[5] “Adaptation through prediction: multisensory active inference torque control”. English. In: *IEEE Transactions on Cognitive and Developmental Systems* 15.1 (2023), pp. 32–41. ISSN: 2379-8920. DOI: 10.1109/TCDS.2022.3156664.

[6] Christian Kampp Kruuse. *A Model-Free Predictive Control Framework for Trajectory Optimisation with a Collaborative Robot*. 2022.

[7] Gabriele Rutigliano. *Visual detection and picking of delicate objects with a soft pneumatic robotic gripper*. 2023.

[8] Yuanxin Xie et al. “An Integrated Multi-Sensor Network for Adaptive Grasping of Fragile Fruits: Design and Feasibility Tests”. In: *Sensors (Basel, Switzerland)* 20 (2020). DOI: 10.3390/s20174973.

[9] A. Mohamed et al. “Soft manipulator robot for selective tomato harvesting”. In: *Precision agriculture ’19* (2019). DOI: 10.3920/978-90-8686-888-9_99.

[10] Christopher L. Buckley et al. “The free energy principle for action and perception: A mathematical review”. In: *Journal of Mathematical Psychology* 81 (2017), pp. 55–79. ISSN: 0022-2496. DOI: https://doi.org/10.1016/j.jmp.2017.09.004. URL: https://www.sciencedirect.com/science/article/pii/S0022249617300962.

[11] Pablo Lanillos et al. *Active Inference in Robotics and Artificial Agents: Survey and Challenges*. 2021. arXiv: 2112.01871 [cs.RO].

[12] Karl Friston et al. “Generalised Filtering”. In: *Mathematical Problems in Engineering* 2010 (Jan. 2010). DOI: 10.1155/2010/621670.

[13] *Pezzato C., Ferrari, R., Henrandez C. Active inference Controller*. 2020. URL: https://github.com/cpezzato/panda_simulation/.

[14] *Pezzato C., Ferrari, R., Henrandez C. Active inference Controller*. 2019. URL: https://github.com/cpezzato/active_inference.

[15] *Soft Gripper Technical Specifications. MODULE B: CORE*. https://thegrippercompany.com/gripper-lips-support/module-b-core/.

[16] *Gabriele Rutigliano, Source code*. 2024. URL: https://github.com/rutiglianogabriele/thesis_uaic_ws.

[17] *Baioumy, M., Pezzato, C., Ferrari, R., & Hawes, N. Unbiased active inference for torque control*. 2021. URL: https://github.com/cpezzato/unbiased_aic.

[18] *libfranka*. https://frankaemika.github.io/docs/libfranka.html.

[19] Yanhao He and Steven Liu. “Analytical Inverse Kinematics for Franka Emika Panda – a Geometrical Solver for 7-DOF Manipulators with Unconventional Design”. In: *2021 9th International Conference on Control, Mechatronics and Automation (ICCMA2021)*. IEEE, Nov. 2021. DOI: 10.1109/ICCMA54375.2021.9646185.

[20] *Intel Realsense, Post-processing filters*. https://dev.intelrealsense.com/docs/post-processing-filters.
[21] Chuxiong Hu et al. “Task Space Contouring Error Estimation and Precision Iterative Control of Robotic Manipulators”. In: *IEEE Robotics and Automation Letters* 7.3 (2022), pp. 7826–7833. DOI: 10.1109/LRA.2022.3180430.
[22] *Franka Emika Panda, robot and interface specifications*. https://frankaemika.github.io/docs/control_parameters.html.
[23] *Franka Emika Panda Specifications*. https://franka.de/documents.

List of Abbreviations

AIC Active Inference Control. 1–3, 5–13, 15, 24–26, 46, 52, 54, 55, 59–61, 63, 65, 66, 69

AIF Active Inference. 6, 12, 69

API Application Programming Interface. 15

DoF Degrees of Freedom. 1, 6, 7, 15

FCI Franka Control Interface. 15, 27, 28, 54, 57

FEP Free Energy Principle. 6

FOV Field of View. 18, 23, 34

GUI Graphical User Interface. 15

IC Impedance Control. 12

IK Inverse Kinematics. 2, 25, 26, 28–30, 63, 65, 66, 69

KPI Key Performance Indicator. 20

LQR Linear Quadratic Gaussian. 67

MAIC Multi-Modal Active Inference Controller. 12

MIR Mobile Industrial Robot. 18

MPC Model Predictive Control. 12

MRAC Model Reference Adaptive Controller. 5, 6, 13, 15, 24, 46, 52, 55, 60, 61, 65, 69

PI Proportional-Integral. 8, 9

RGB Red Green Blue. 17, 32, 34

RMS Root Mean Square. 44, 46, 52, 65

ROS Robot Operating System. 2, 15, 17, 22, 24, 26, 64

RPA Robot Process Automation. 1

uAIC Unbiased Active Inference Control. 1–3, 5, 6, 8–13, 20, 21, 24–26, 28–30, 39, 46, 52, 54, 55, 60, 61, 63–67, 69

VSCode Visual Studio Code. 16

A Franka Emika Panda Limits

Name	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6	Joint 7	Unit
q_{max}	2.8973	1.7628	2.8973	-0.0698	2.8973	3.7525	2.8973	rad
q_{min}	-2.8973	-1.7628	-2.8973	-3.0718	-2.8973	-0.0175	-2.8973	rad
$\dot{q}_{max}$	2.1750	2.1750	2.1750	2.1750	2.6100	2.6100	2.6100	$\frac{rad}{s}$
$\ddot{q}_{max}$	15	7.5	10	12.5	15	20	20	$\frac{rad}{s^2}$
$\dddot{q}_{max}$	7500	3750	5000	6250	7500	10000	10000	$\frac{rad}{s^3}$
$\tau_{j_{max}}$	87	87	87	87	12	12	12	Nm
$\dot{\tau}_{j_{max}}$	1000	1000	1000	1000	1000	1000	1000	$\frac{Nm}{s}$

Table A.1: Franka Emika Panda joints limits [22]

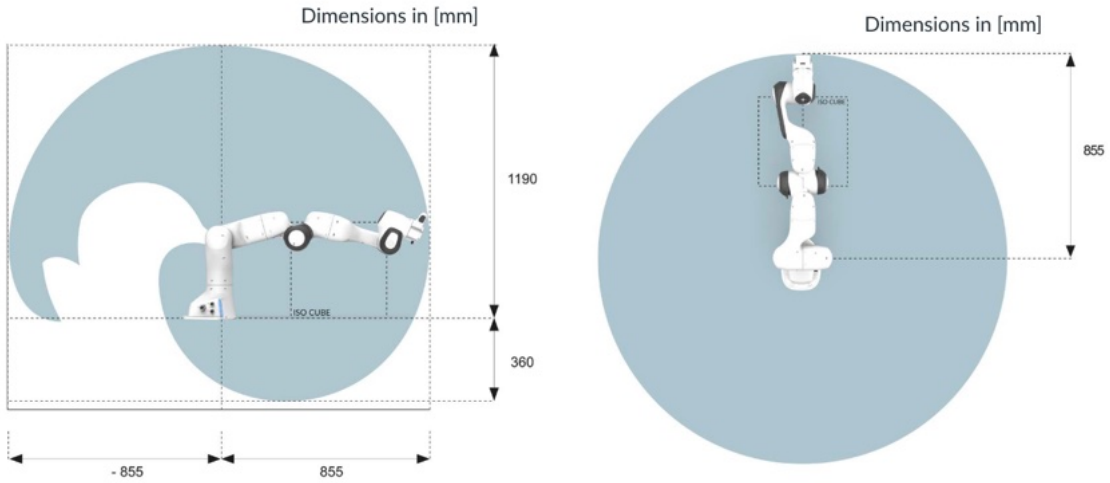


Figure A.1: Reachable space of the Franka Emika Panda robot [23]

B Tuning Parameters for the MRAC and AIC

B.0.1 AIC

	Joint Number						
	1	2	3	4	5	6	7
SigmaP_mu	1/15.0	1/15.0	1/15.0	1/15.0	0.1/15.0	0.05/15.0	0.01/15.0
SigmaP_muprime	1/30.0	1/30.0	1/30.0	1/30.0	0.1/30.0	0.05/30.0	0.01/30.0
SigmaP_yq0	1	1	1	1	1	1	1
SigmaP_yq1	1	1	1	1	1	1	1
k_mu	11.67	11.67	11.67	11.67	11.67	11.67	11.67
k_a	200	200	200	200	200	200	200

Table B.1: Table of the tuning parameters used for the AIC in this project

B.0.2 MRAC

	Joint Number						
	1	2	3	4	5	6	7
omega	2	2	2	2	1	1	1
zeta	1	1	1	1	1	1	1
alpha1	0.2	0.1	0.2	0.5	1	2	2
alpha2	0.1	0.5	0.1	3	0.01	0.1	0.1
E01	2	2	2	2	2	2	1
E02	0.02	5	0.2	10	0.2	0.2	0.1
E11	0.001	0.001	0.001	0.001	0.001	0.001	0.001
E12	0	0	0	0	0	0	0
F01	5.105	5.105	5.105	0.105	0.105	0.105	0.105
F02	0	0	0	0	0	0	0
F11	0.0001	0.0001	0.0001	0.0001	0.00001	0.00001	0.00001
F12	0	0	0	0	0	0	0
l1	1	1	1	1	1	1	1
l2	1	1	1	1	1	1	1

Table B.2: Table of the tuning parameters used for the MRAC in this project

C Algorithms Implemented on the E-Grocery Tasks

C.1 Grasping Static Object from Unknown Locations

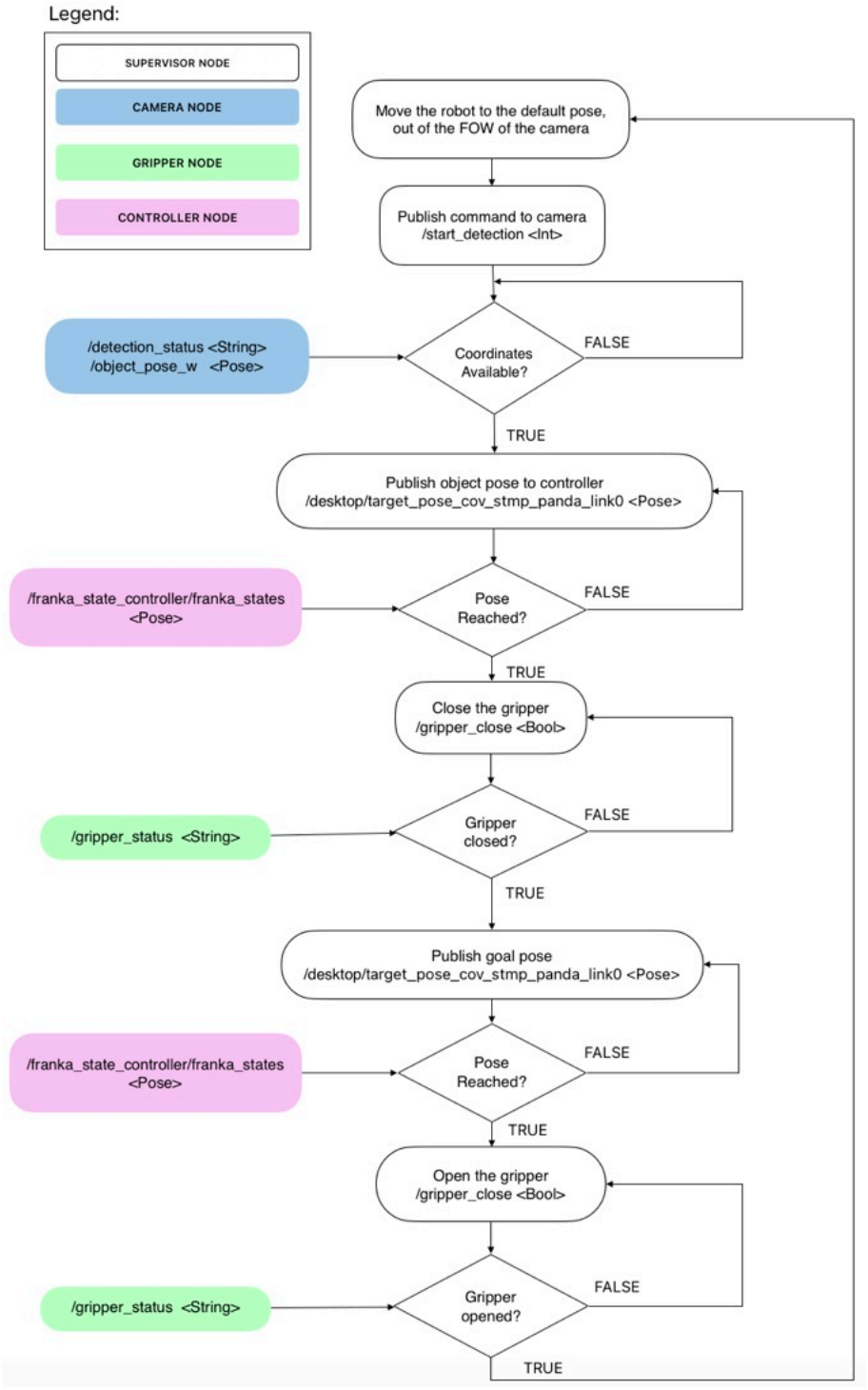


Figure C.1: Decision diagram for picking static objects from unknown locations

C.2 Grasping Moving Object from Unknown Locations

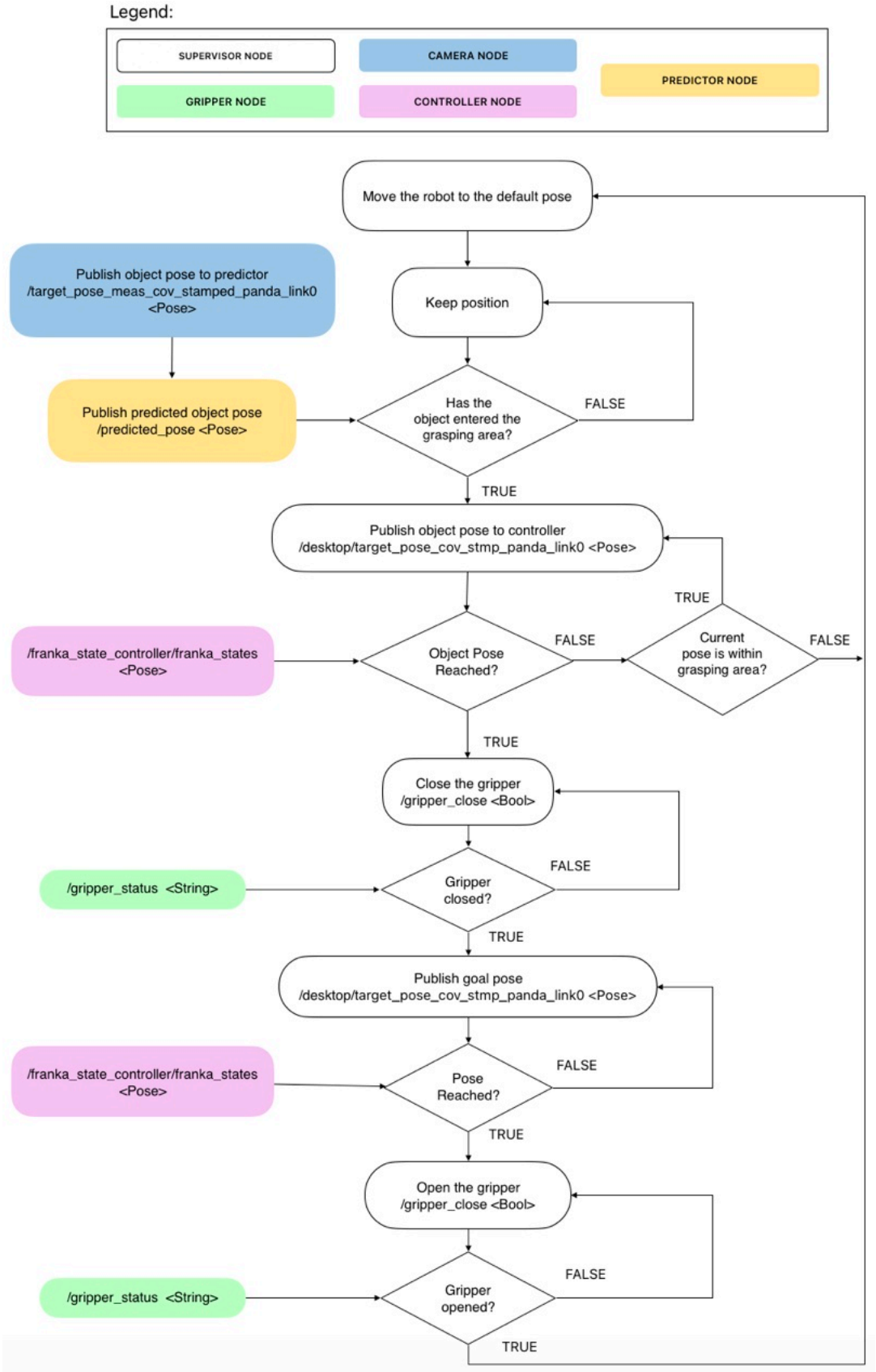


Figure C.2: Decision diagram for picking moving objects from unknown locations

Axis	X	Y	Z	Unit
Threshold	0.03	0.03	0.04	m

Table C.1: Thresholds used when grasping static objects

Axis	X	Y	Z	Unit
Threshold	0.03	0.075	0.05	m

Table C.2: Thresholds used when grasping moving objects

D Tuning of the uAIC

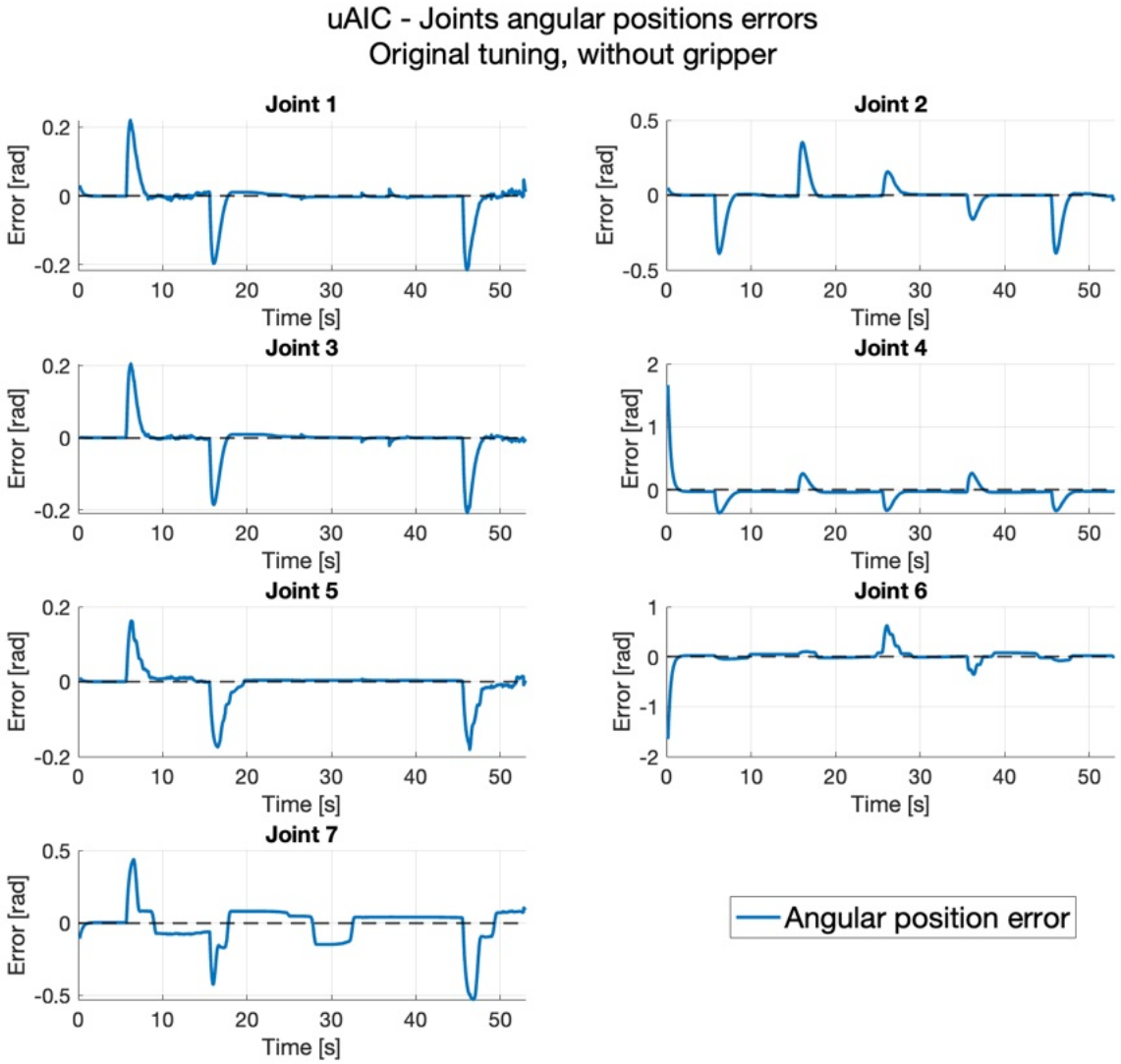


Figure D.1: Joint-space tracking errors of the uAIC when tuned with the original tuning parameters and the gripper detached, while navigating through static waypoints. For each joint, the difference between the desired and measured joint angular position is shown.

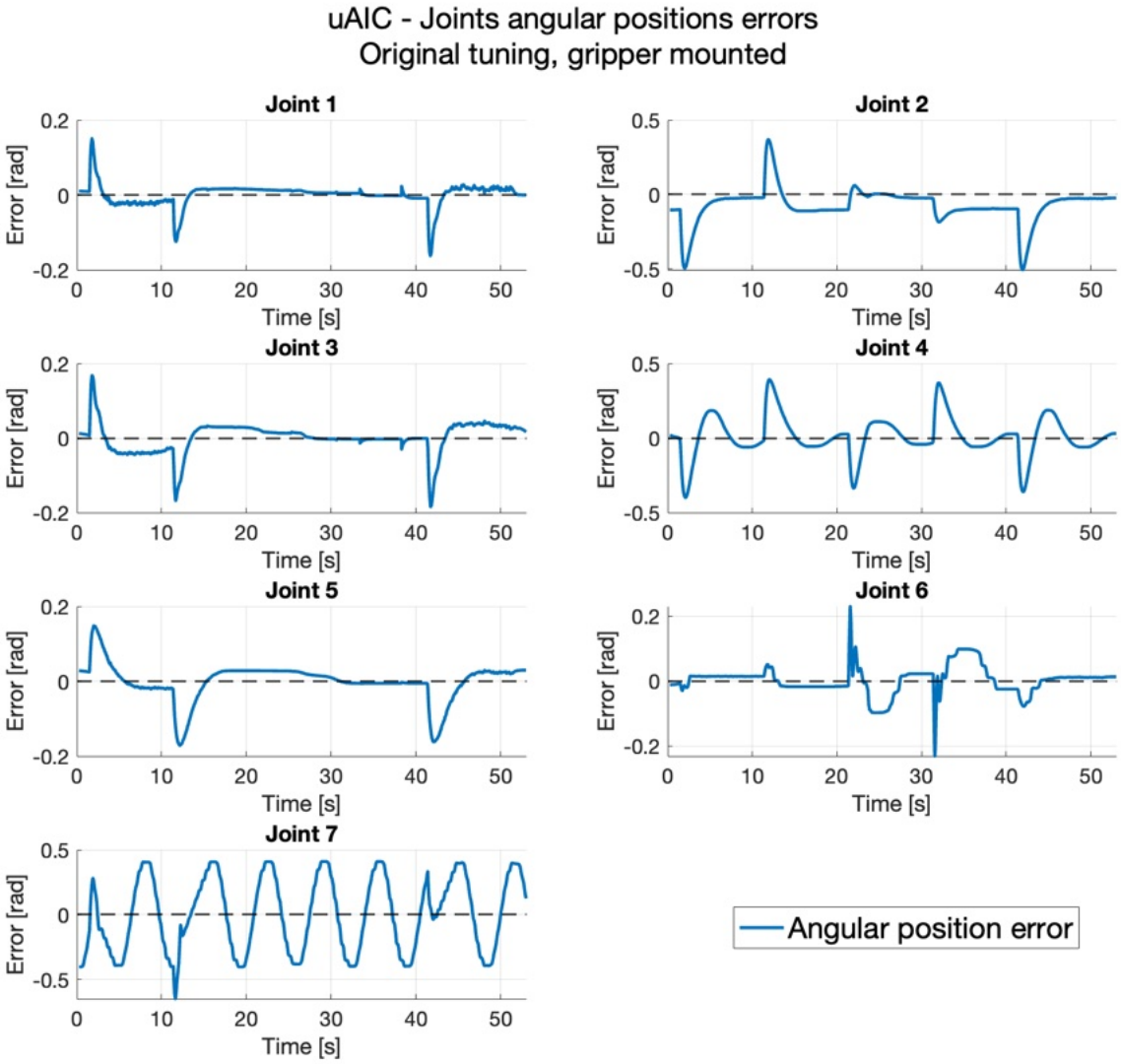


Figure D.2: Joint-space tracking errors of the uAIC when tuned with the original tuning parameters and the gripper attached, while navigating through static waypoints. For each joint, the difference between the desired and measured joint angular position is shown.

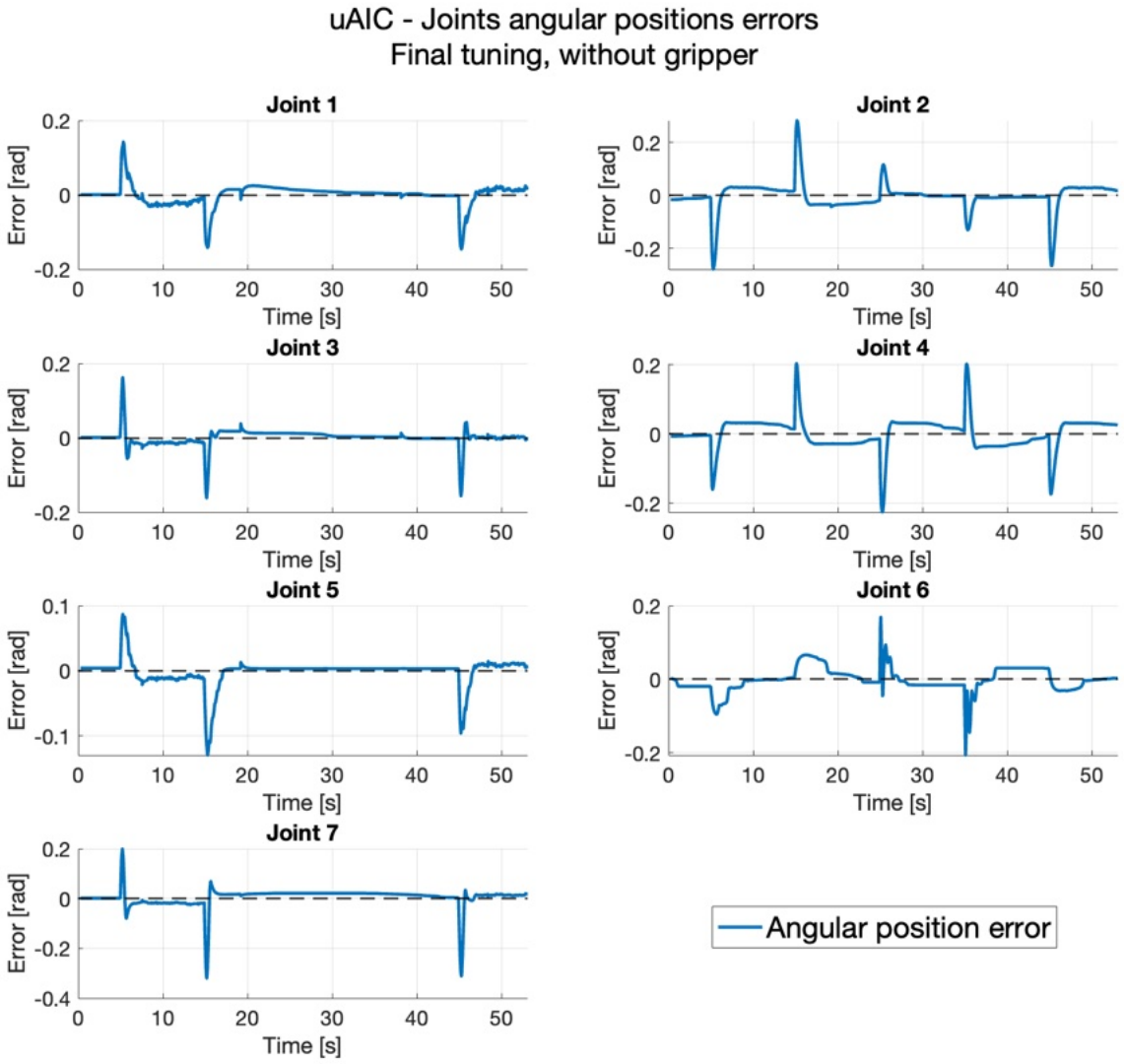


Figure D.3: Joint-space tracking errors of the uAIC when tuned with the final tuning parameters and the gripper detached, while navigating through static waypoints. For each joint, the difference between the desired and measured joint angular position is shown.

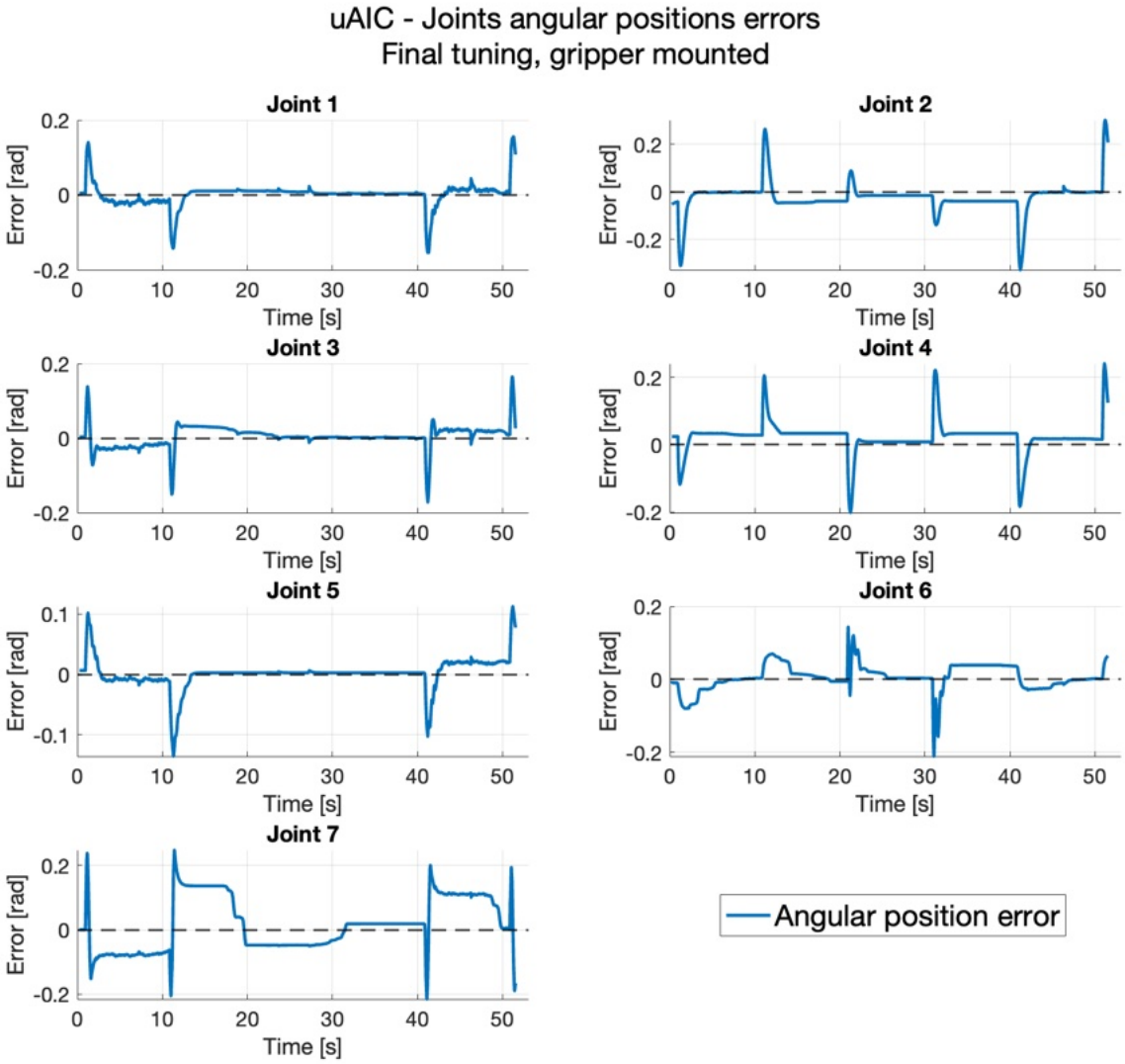


Figure D.4: Joint-space tracking errors of the uAIC when tuned with the final tuning parameters and the gripper attached, while navigating through static waypoints. For each joint, the difference between the desired and measured joint angular position is shown.

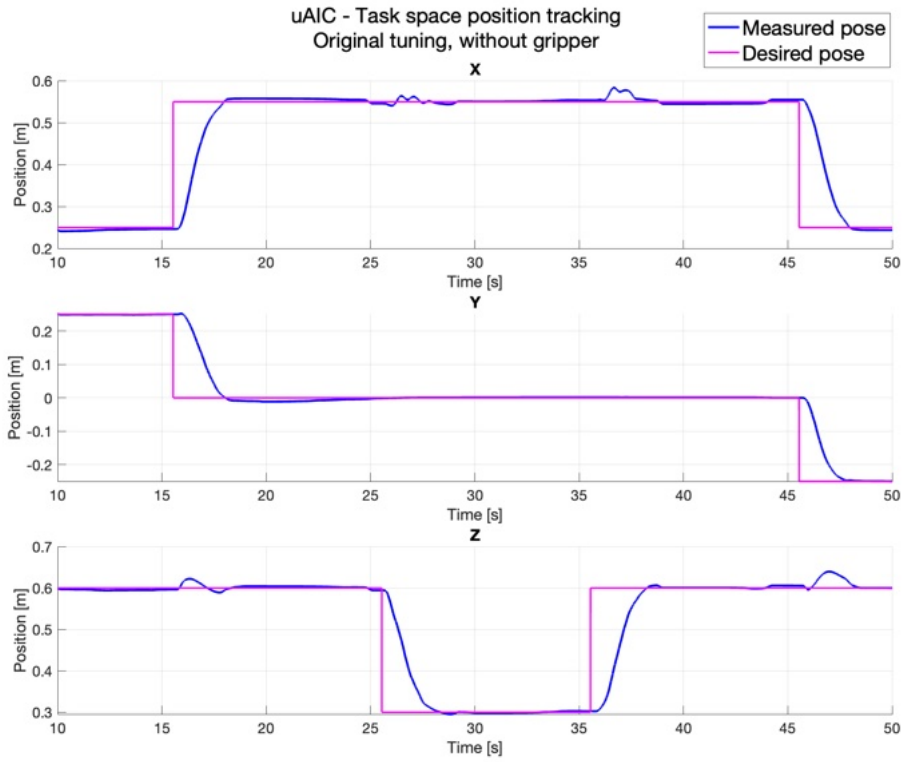


Figure D.5: Joint-space tracking performance of the uAIC when tuned with the original tuning parameters and the gripper detached, while navigating through static waypoints. The actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom).

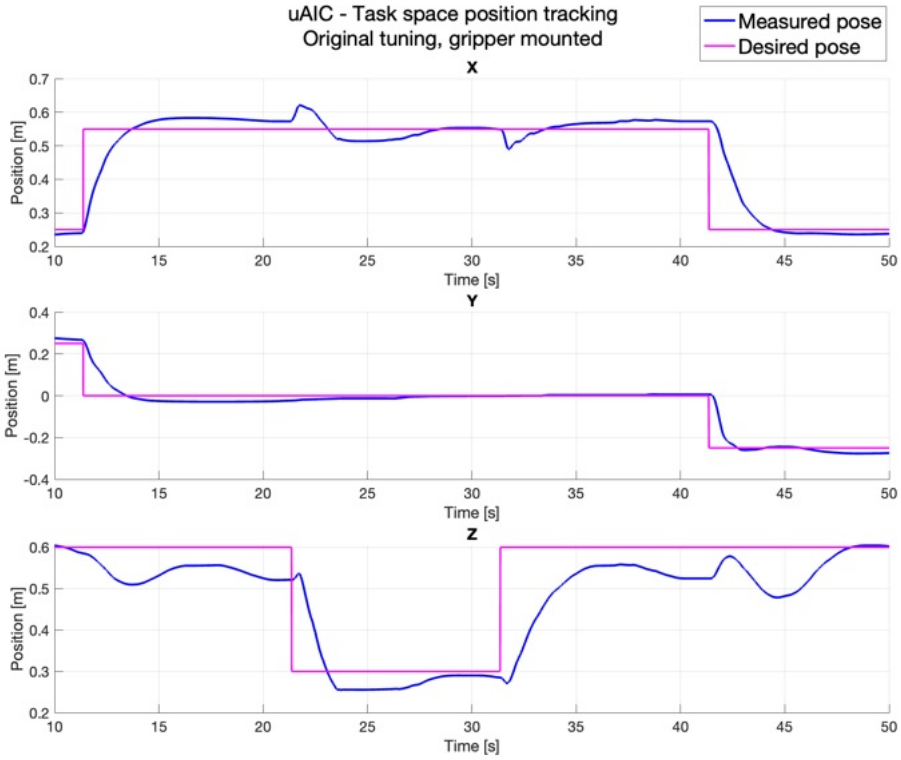


Figure D.6: Joint-space tracking performance of the uAIC when tuned with the original tuning parameters and the gripper attached, while navigating through static waypoints.The actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom).

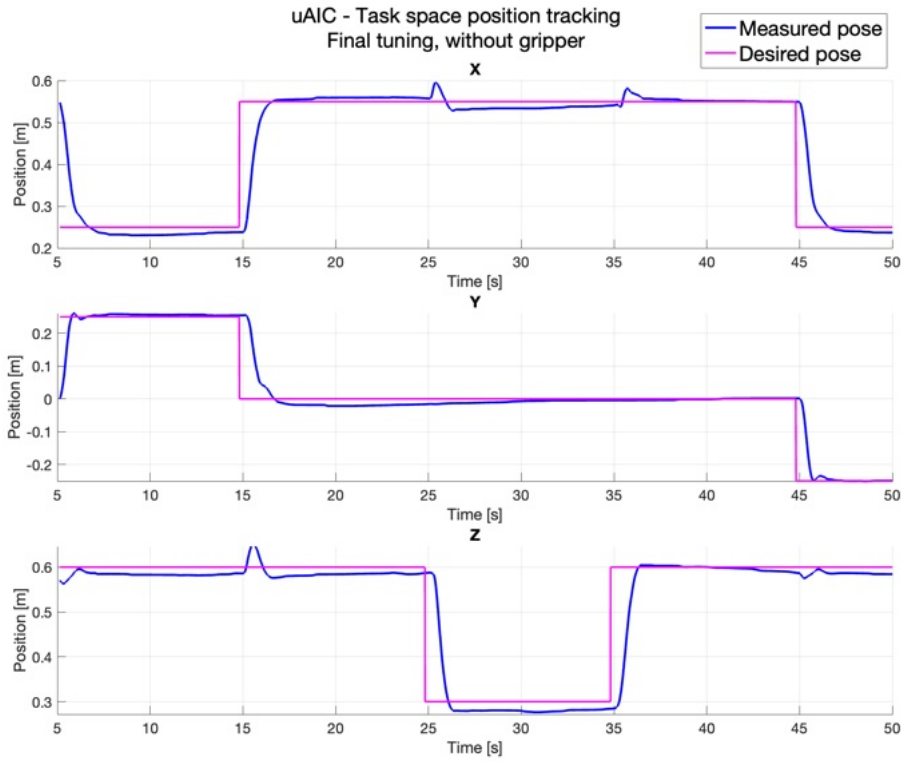


Figure D.7: Joint-space tracking performance of the uAIC when tuned with the final tuning parameters and the gripper detached, while navigating through static waypoints. The actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom).

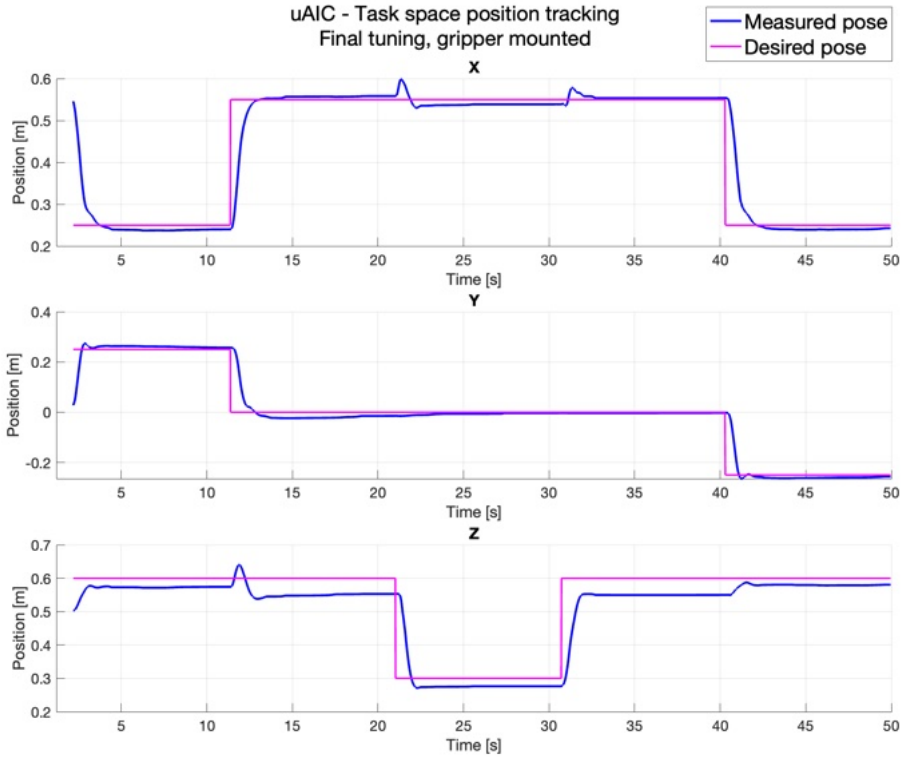


Figure D.8: Joint-space tracking performance of the uAIC when tuned with the final tuning parameters and the gripper detached, while navigating through static waypoints. The actual end-effector pose (in — blue), as measured by the *libfranka* API, is compared against the desired task-space pose (in — magenta), for the X, Y and Z axes (from top to bottom).

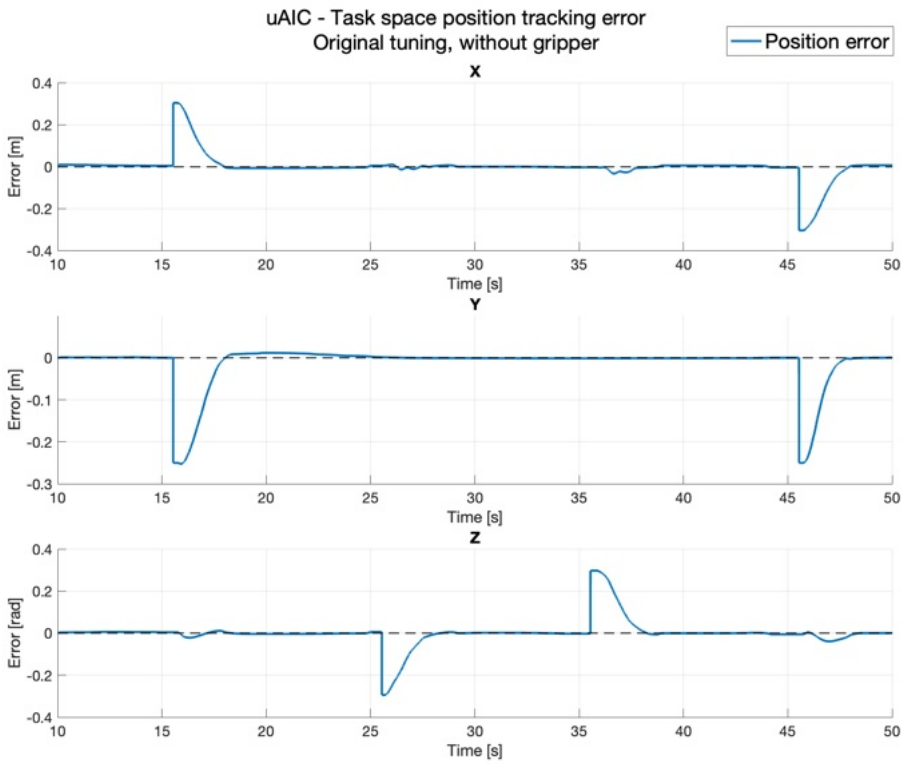


Figure D.9: Task-space tracking errors of the uAIC when tuned with the original tuning parameters and the gripper detached, while navigating through static waypoints.

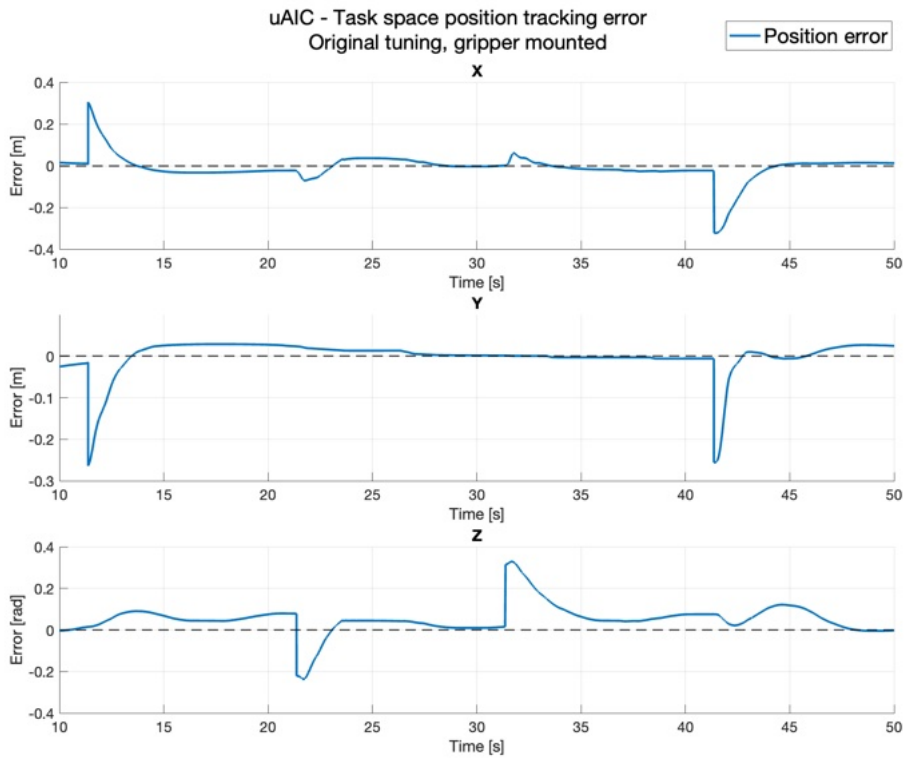


Figure D.10: Task-space tracking errors of the uAIC when tuned with the original tuning parameters and the gripper attached, while navigating through static waypoints.

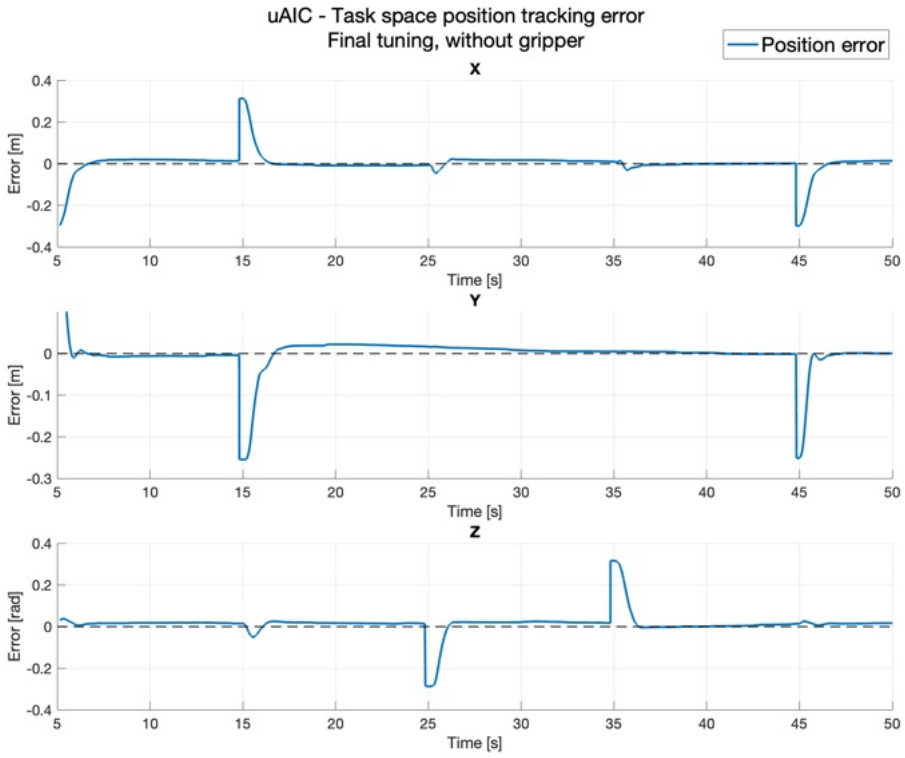


Figure D.11: Task-space tracking errors of the uAIC when tuned with the final tuning parameters and the gripper detached, while navigating through static waypoints.

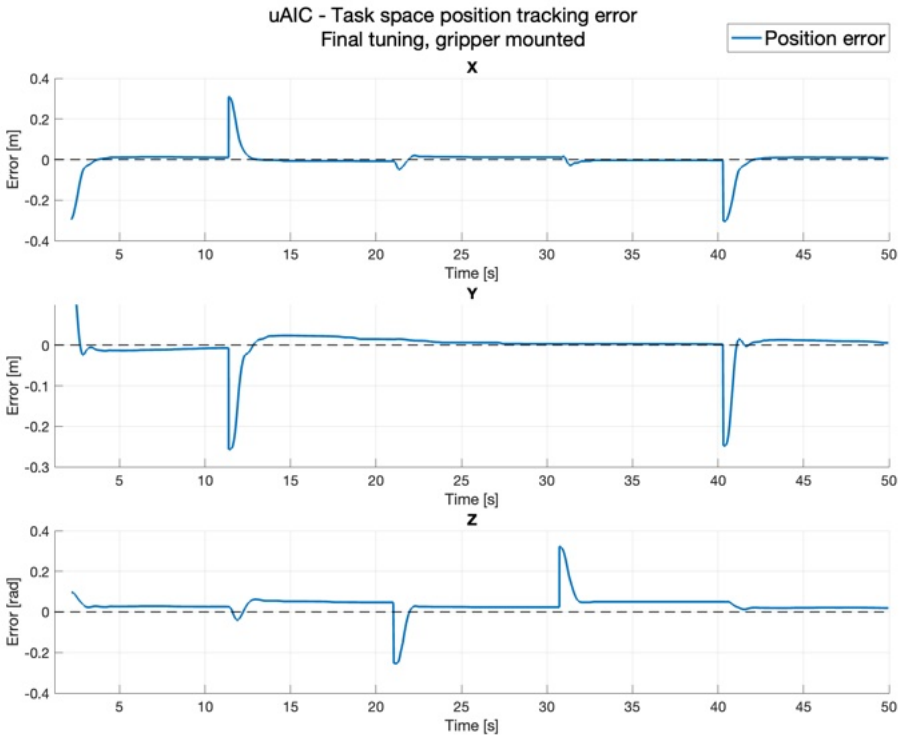


Figure D.12: Task-space tracking errors of the uAIC when tuned with the final tuning parameters and the gripper attached, while navigating through static waypoints.

E Experimental results

E.1 Comparison of Tracking Performances

E.1.1 Waypoints Navigation

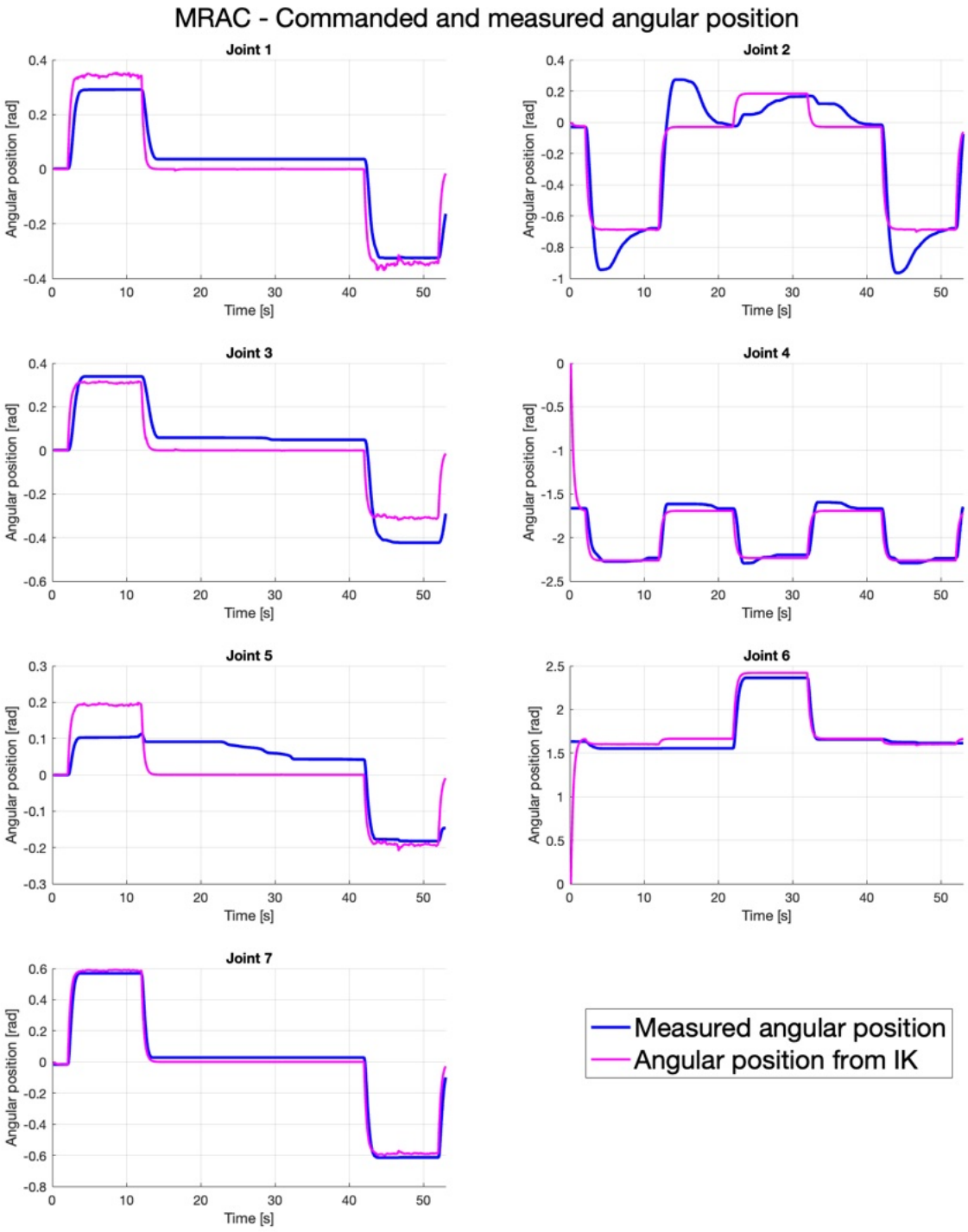


Figure E.1: Joint-space tracking of the MRAC when navigating through static waypoints. For each joint, the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

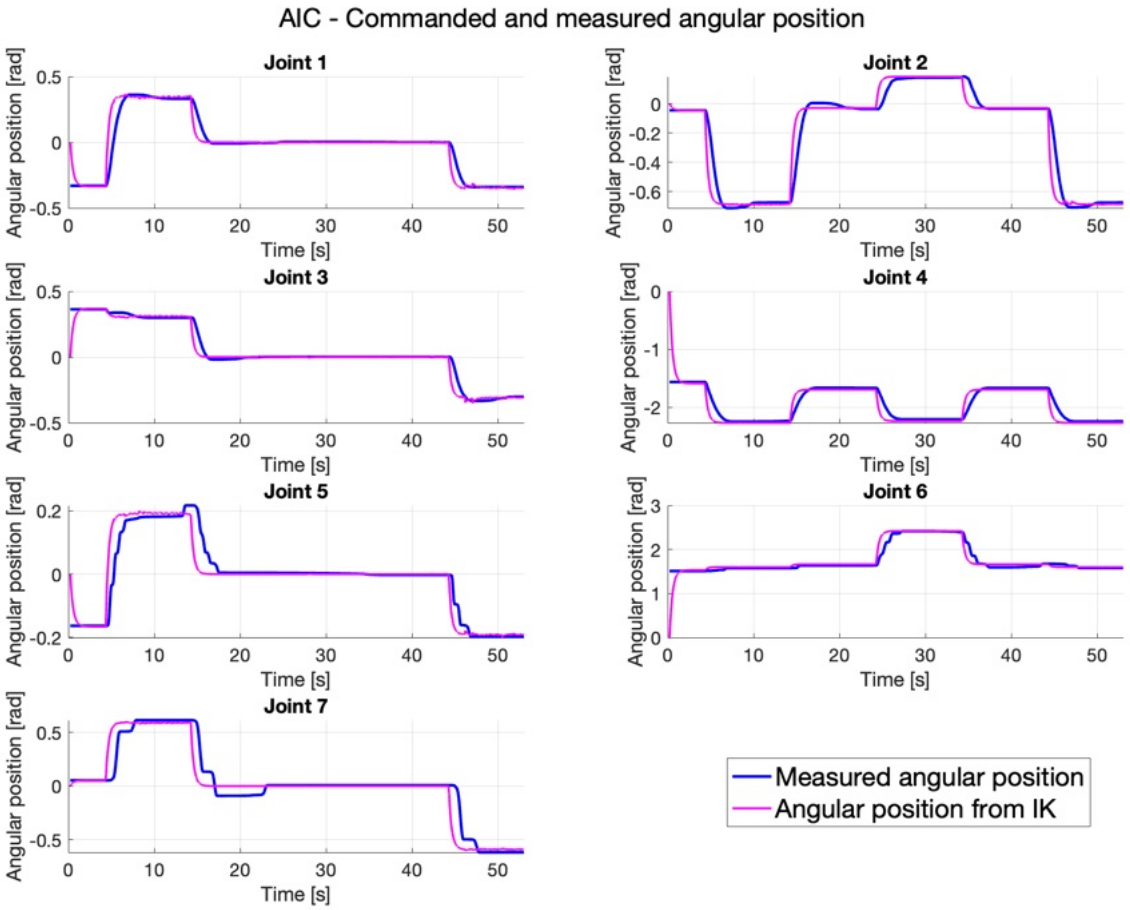


Figure E.2: Joint-space tracking of the AIC when navigating through static waypoints. For each joint, the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

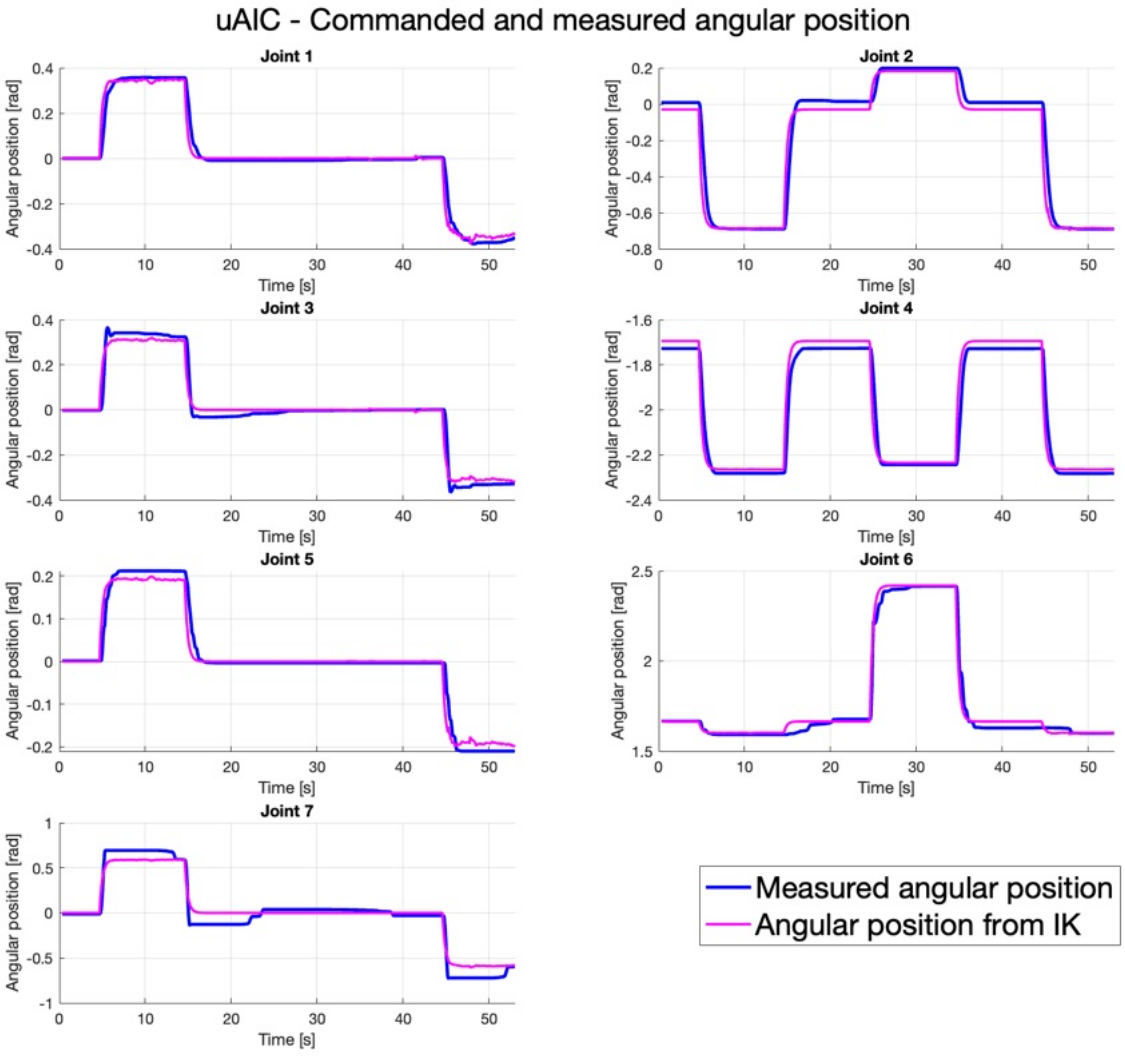


Figure E.3: Joint-space tracking of the uAIC when navigating through static waypoints. For each joint, the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

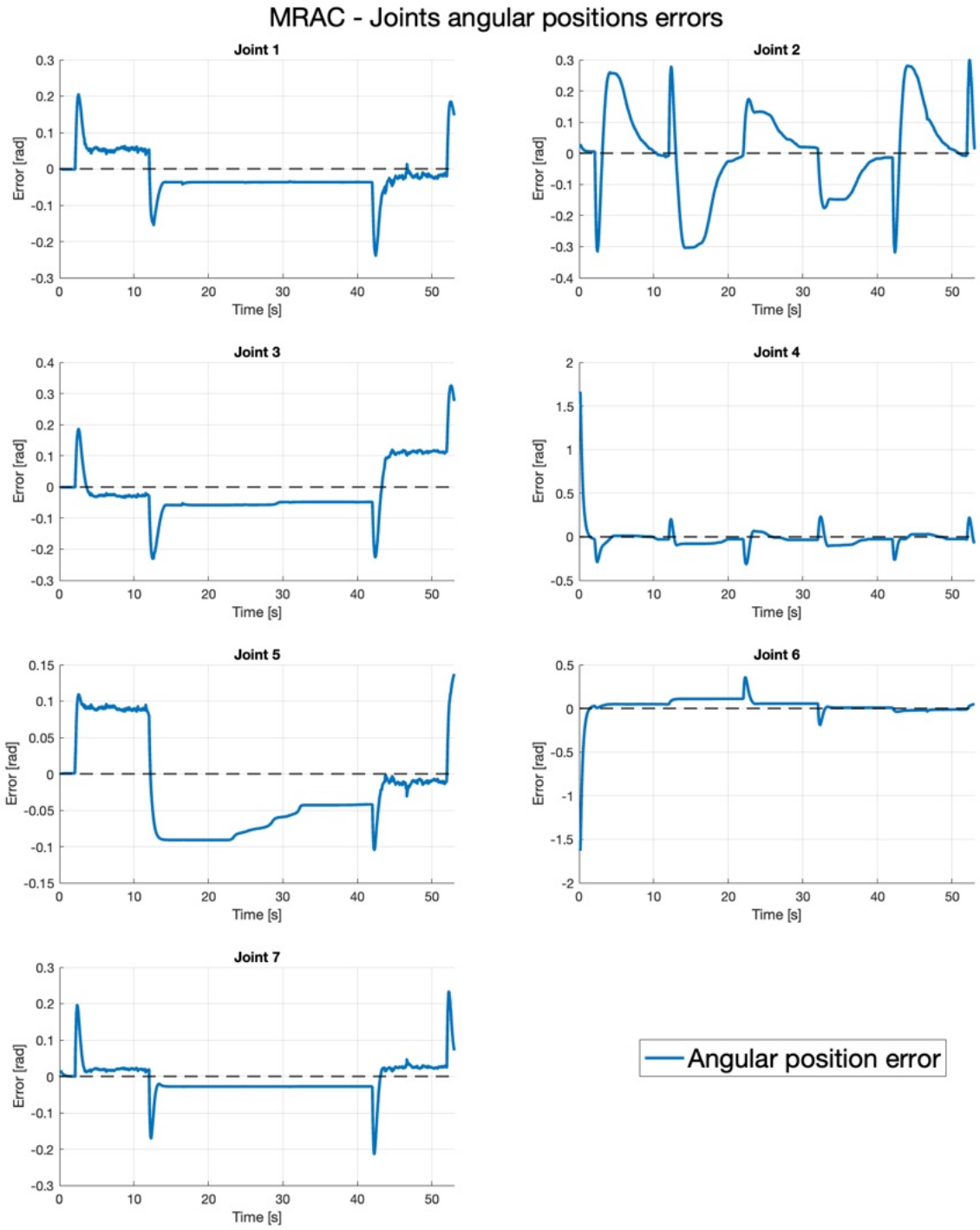


Figure E.4: Joint-space tracking errors of the MRAC when navigating through static waypoints. For each joint, the difference between the desired and measured joint angular position is shown.

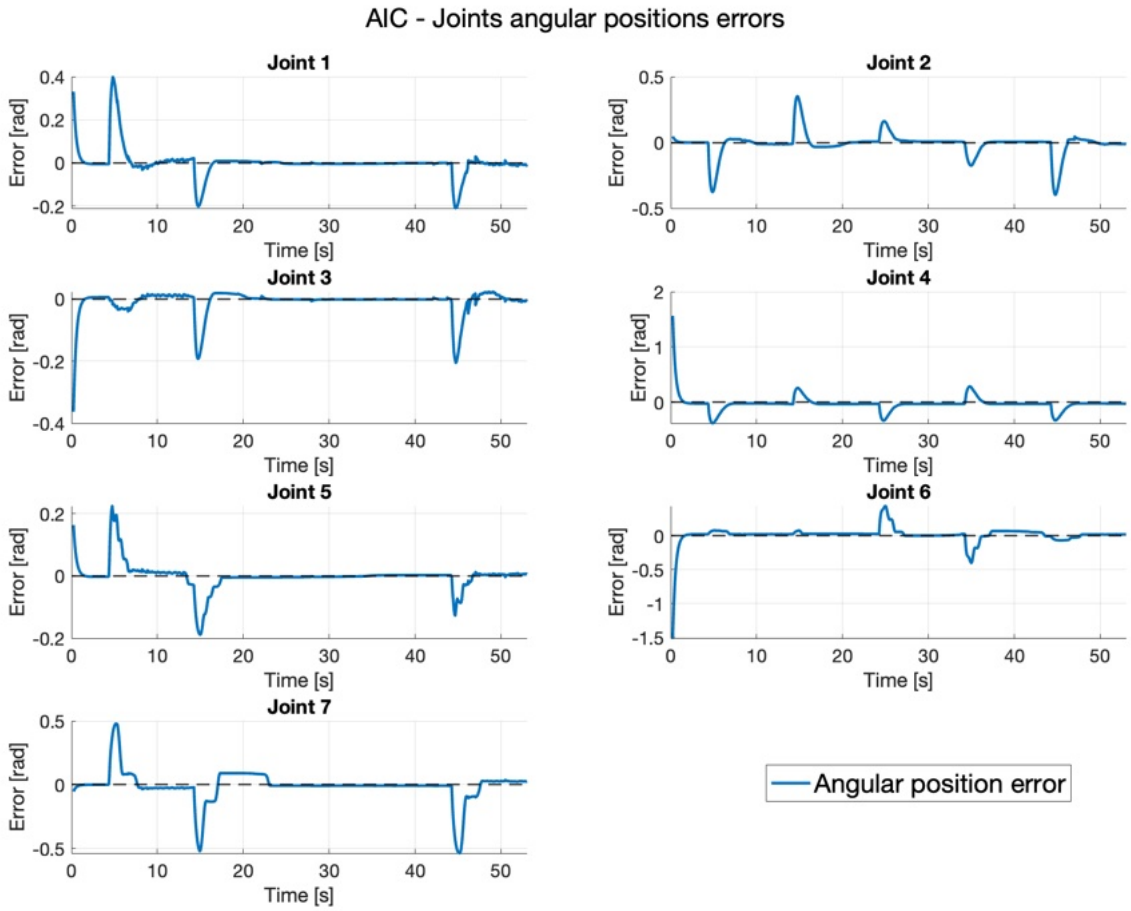


Figure E.5: Joint-space tracking errors of the AIC when navigating through static waypoints. For each joint, the difference between the desired and measured joint angular position is shown.

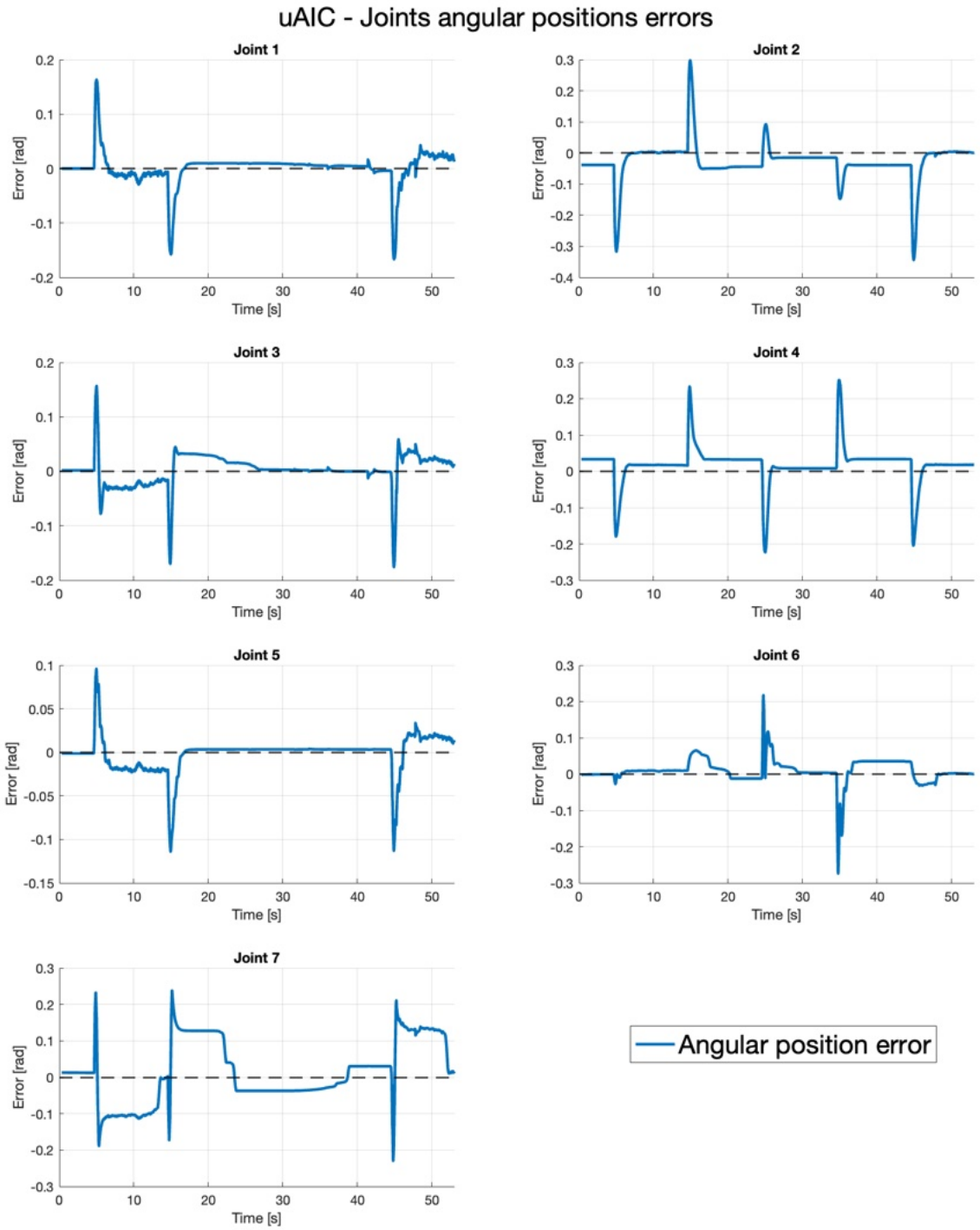


Figure E.6: Joint-space tracking errors of the uAIC when navigating through static waypoints. For each joint, the difference between the desired and measured joint angular position is shown.

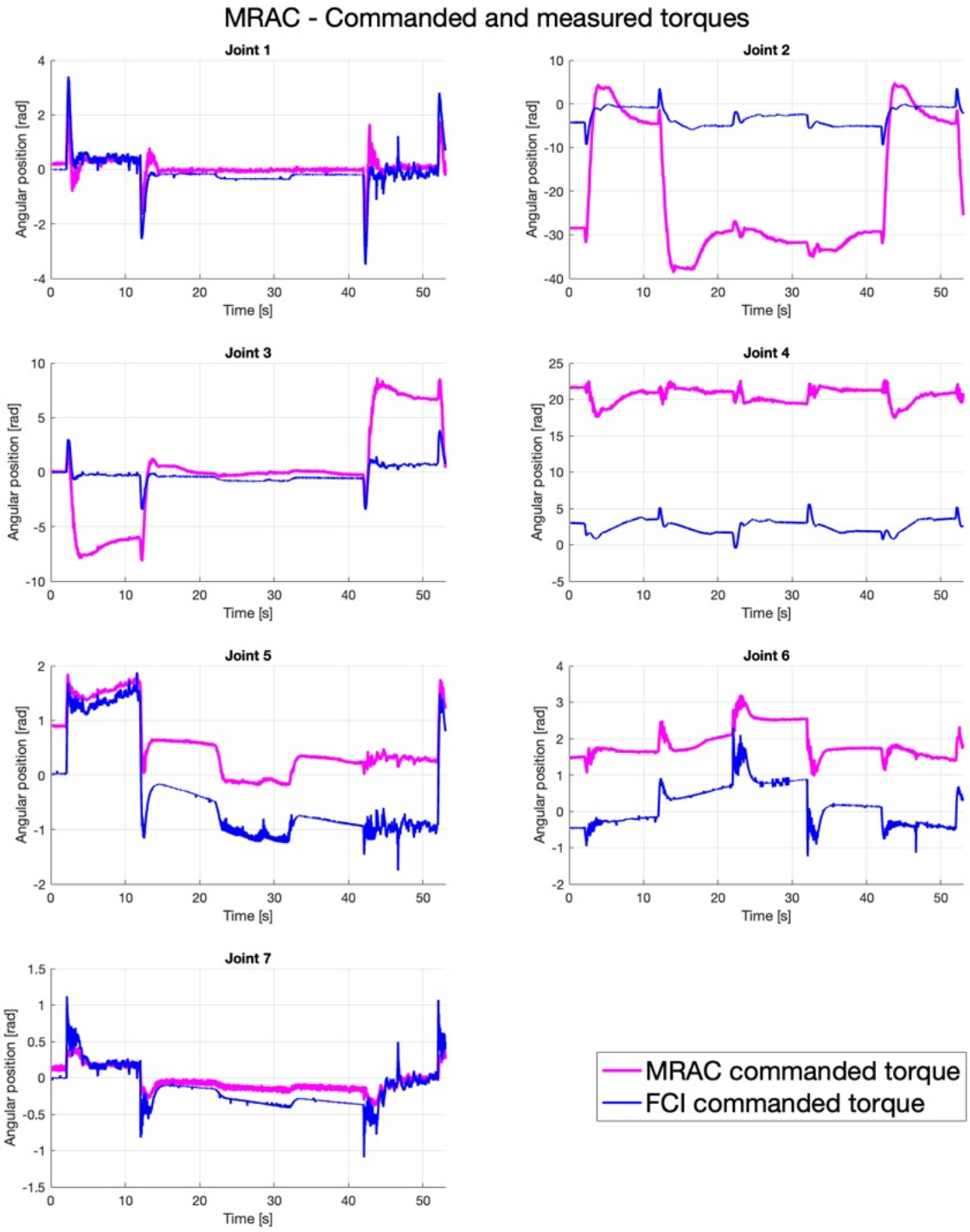


Figure E.7: Torque dynamics when navigating through static waypoints. For each joint, the MRAC torque commands (in — magenta) are compared against the actual torque applied to the joint by the FCI (in — blue).

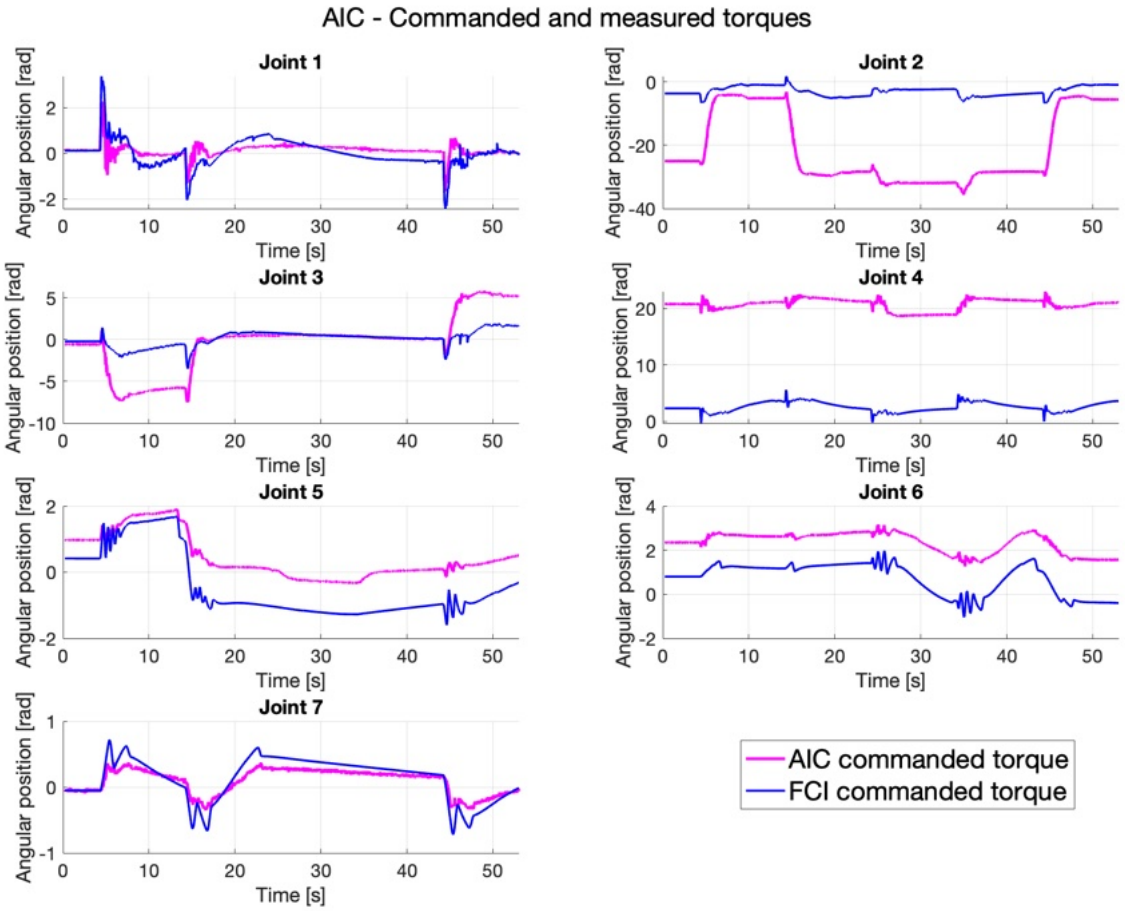


Figure E.8: Torque dynamics when navigating through static waypoints. For each joint, the AIC torque commands (in — magenta) are compared against the actual torque applied to the joint by the FCI (in — blue).

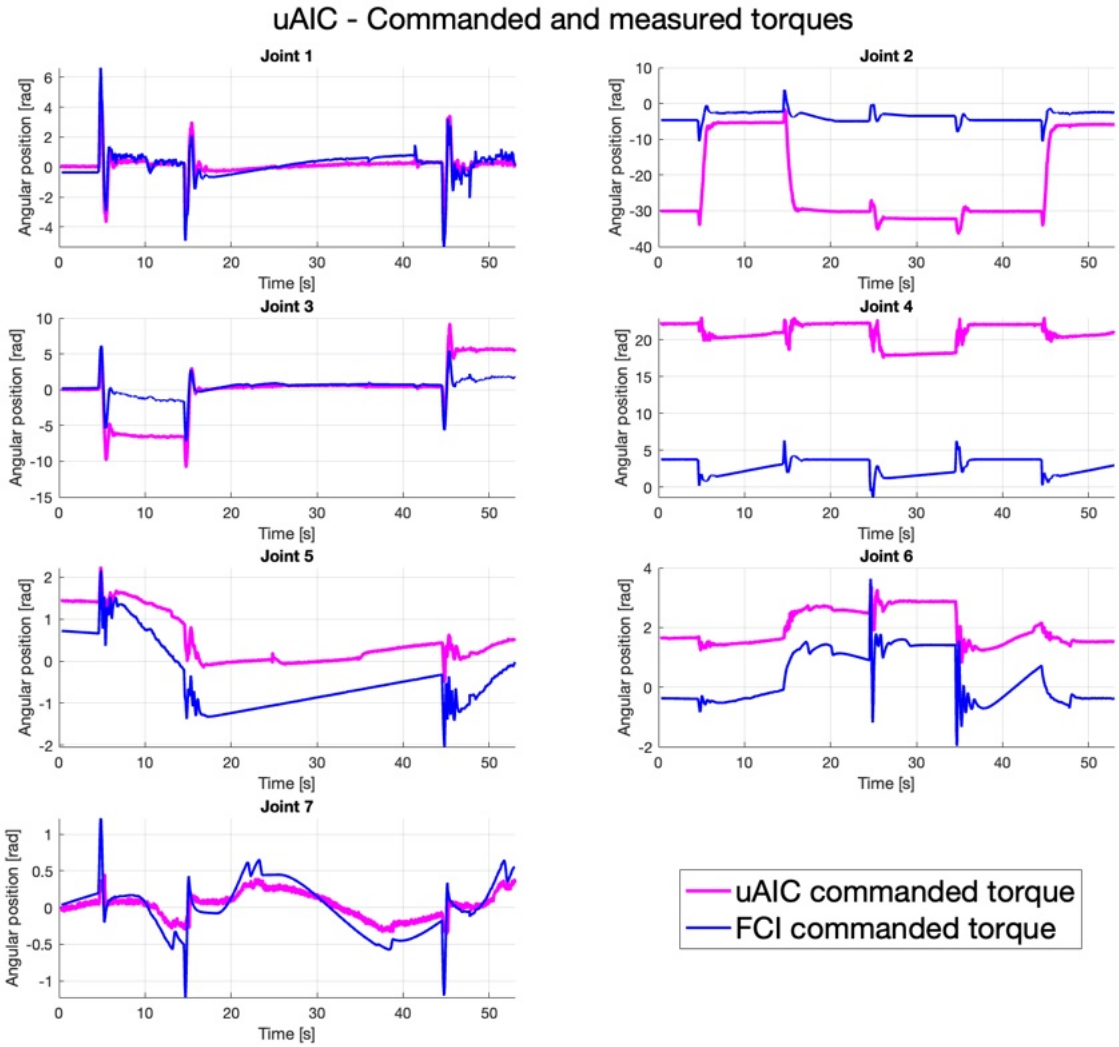


Figure E.9: Torque dynamics when navigating through static waypoints. For each joint, the uAIC torque commands (in — magenta) are compared against the actual torque applied to the joint by the FCI (in — blue).

E.1.2 Tracking of a Dynamic Reference

E.1.2.1 Speed 1000

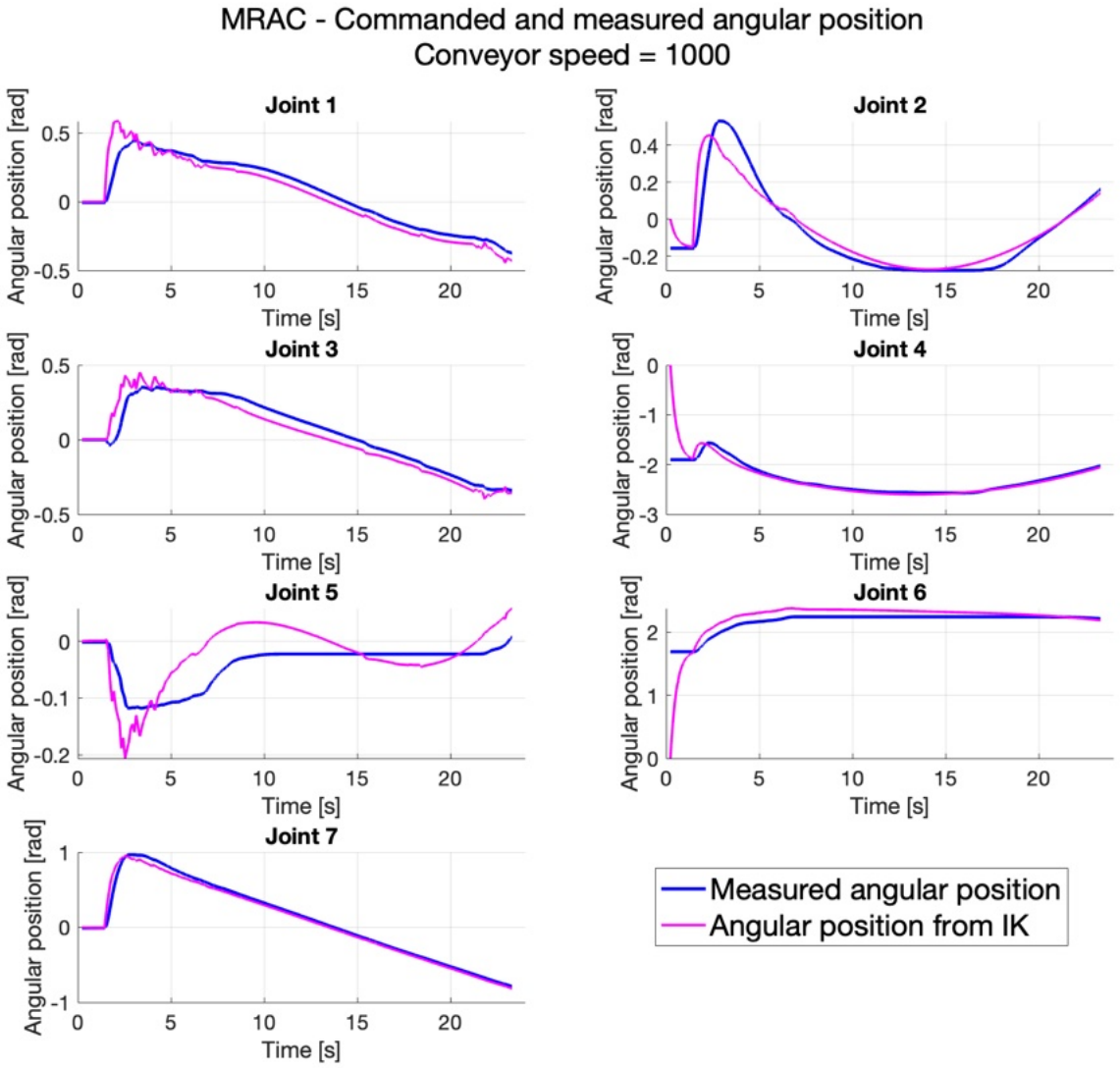


Figure E.10: Joint-space tracking of the MRAC at a conveyor belt speed of 1000. For each joint, the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

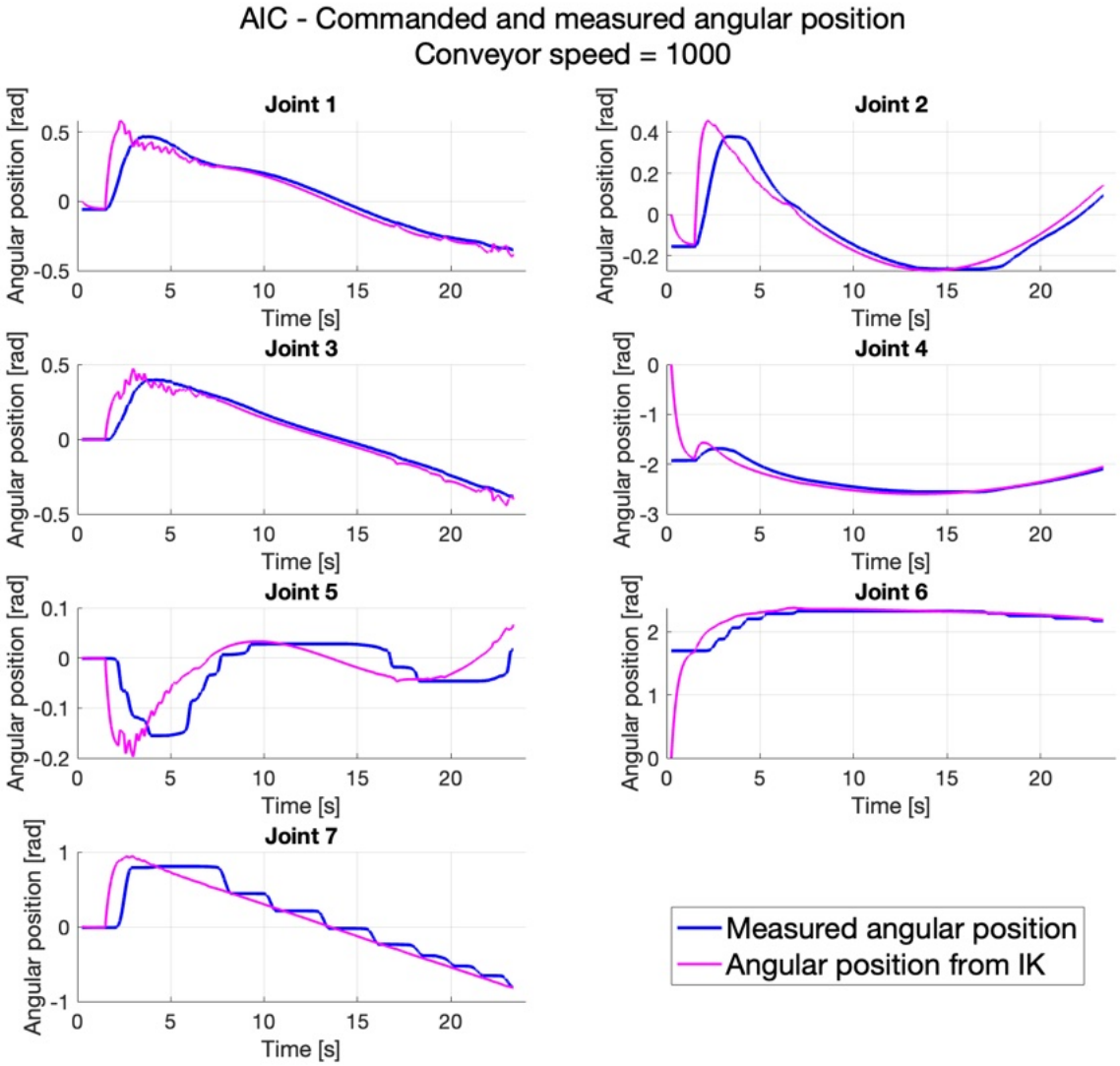


Figure E.11: Joint-space tracking of the AIC at a conveyor belt speed of 1000. For each joint, the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

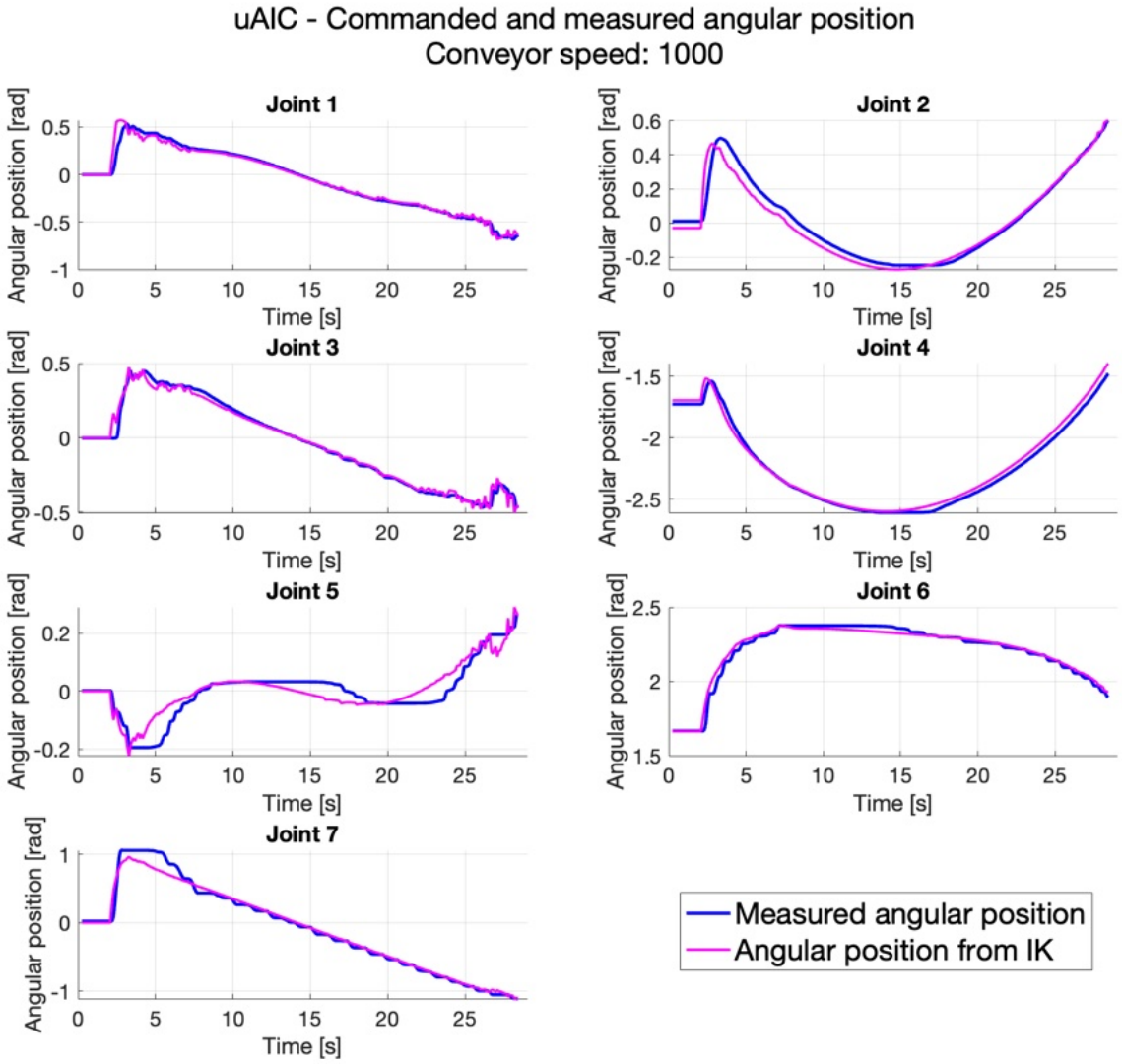


Figure E.12: Joint-space tracking of the uAIC at a conveyor belt speed of 1000. For each joint, the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

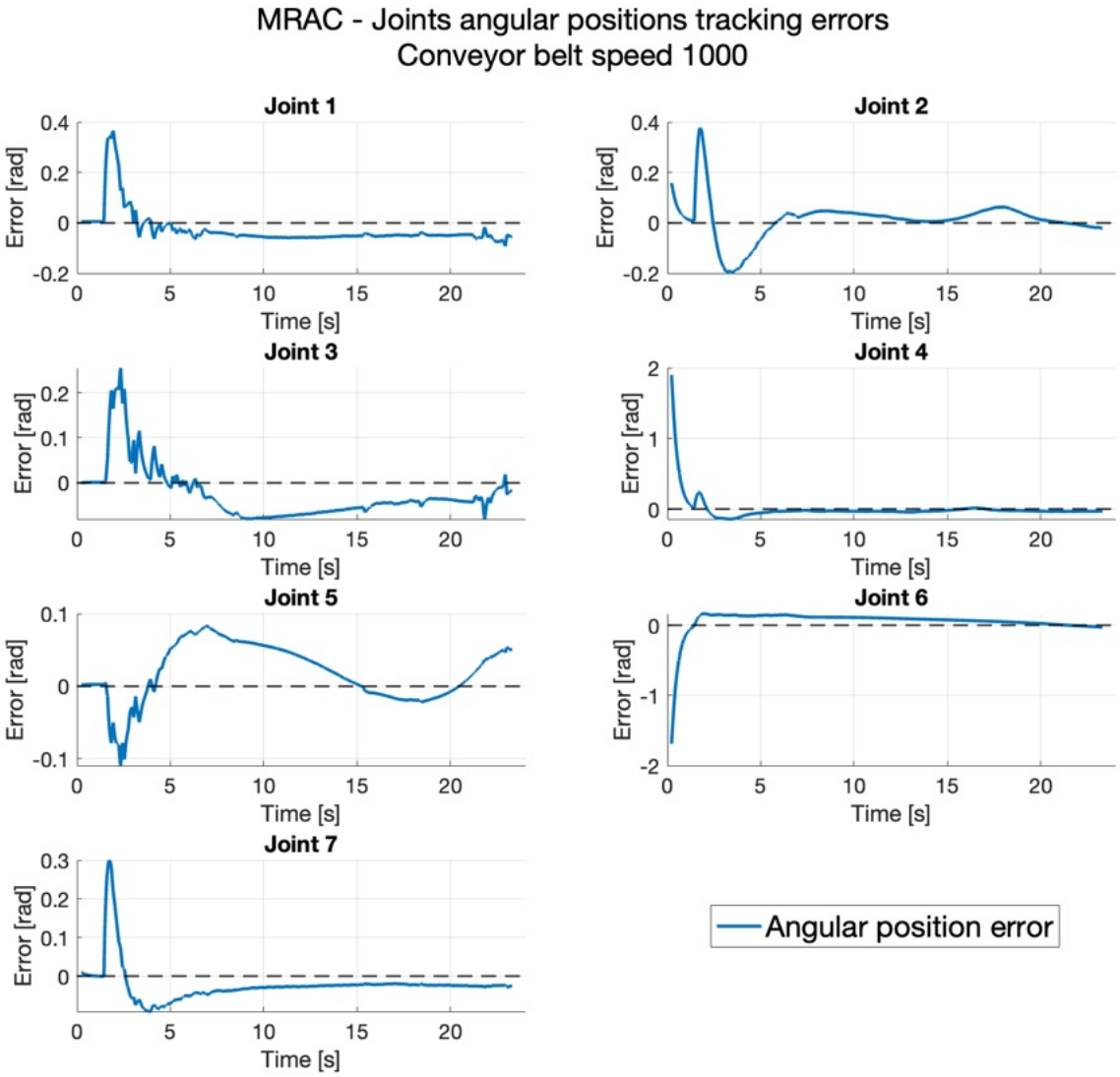


Figure E.13: Joint-space tracking error of the MRAC at a conveyor belt speed of 1000. For each joint, the difference between the desired and measured joint angular position is shown.

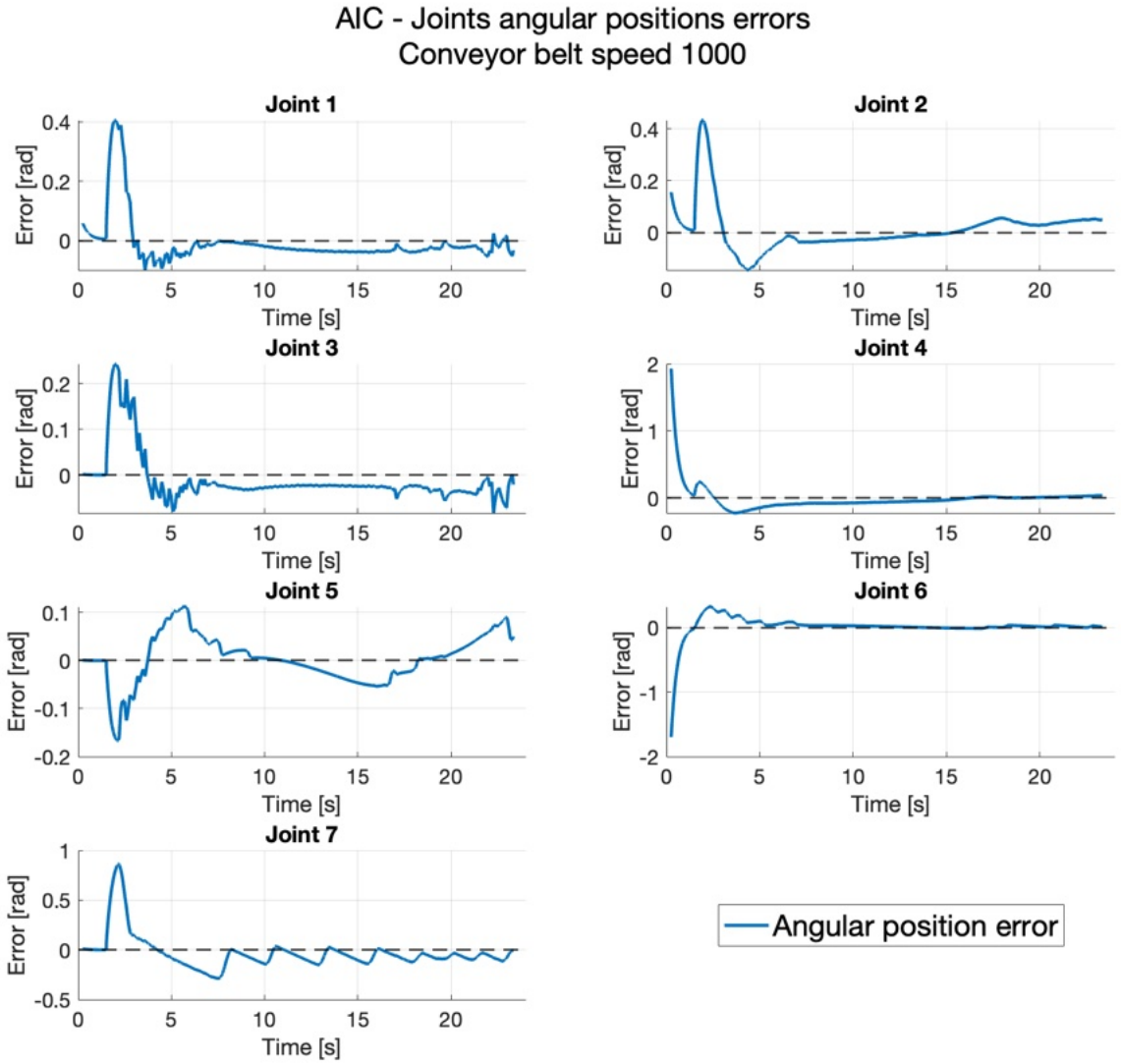


Figure E.14: Joint-space tracking error of the AIC at a conveyor belt speed of 1000. For each joint, the difference between the desired and measured joint angular position is shown.

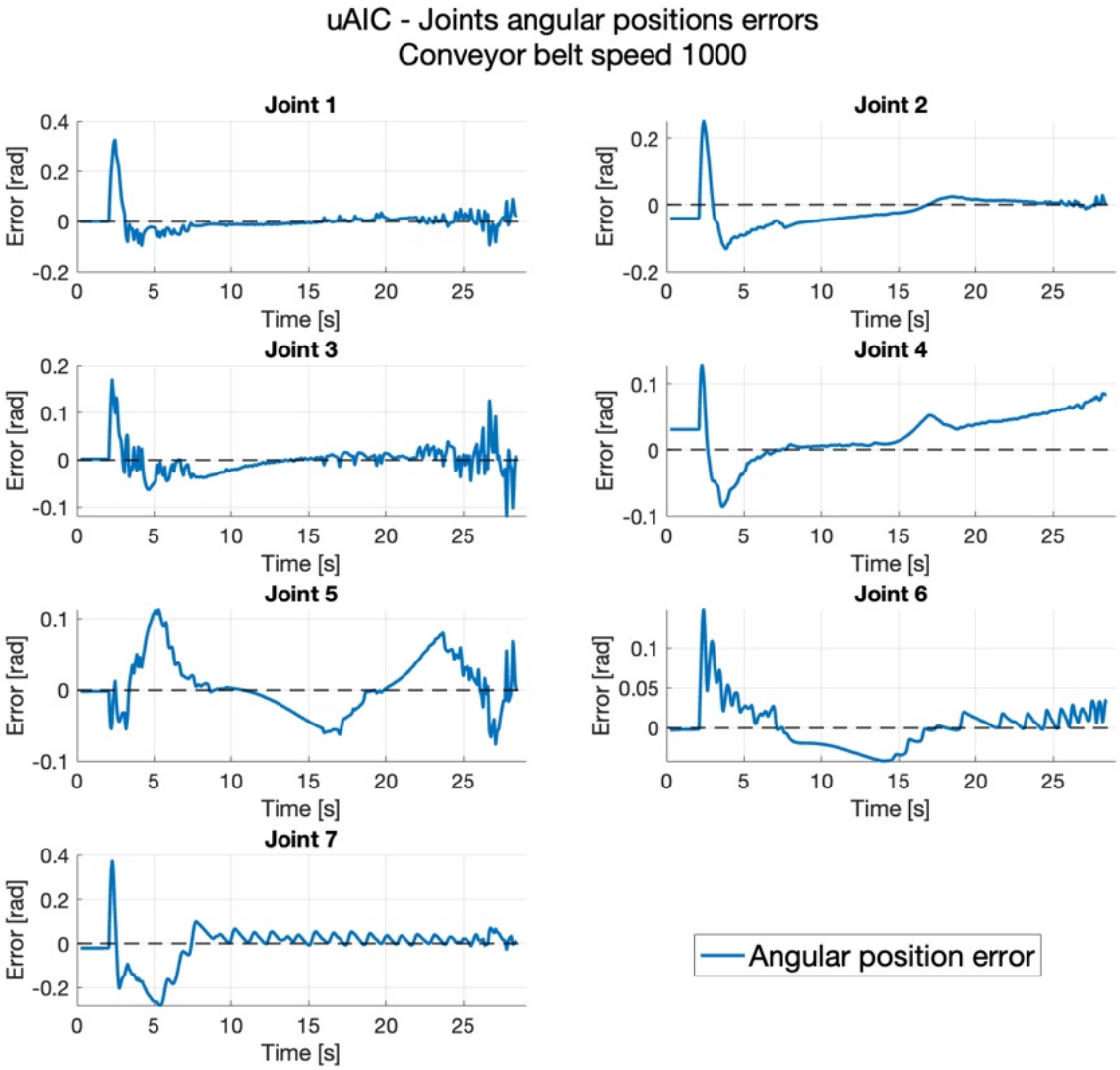


Figure E.15: Joint-space tracking error of the uAIC at a conveyor belt speed of 1000. For each joint, the difference between the desired and measured joint angular position is shown.

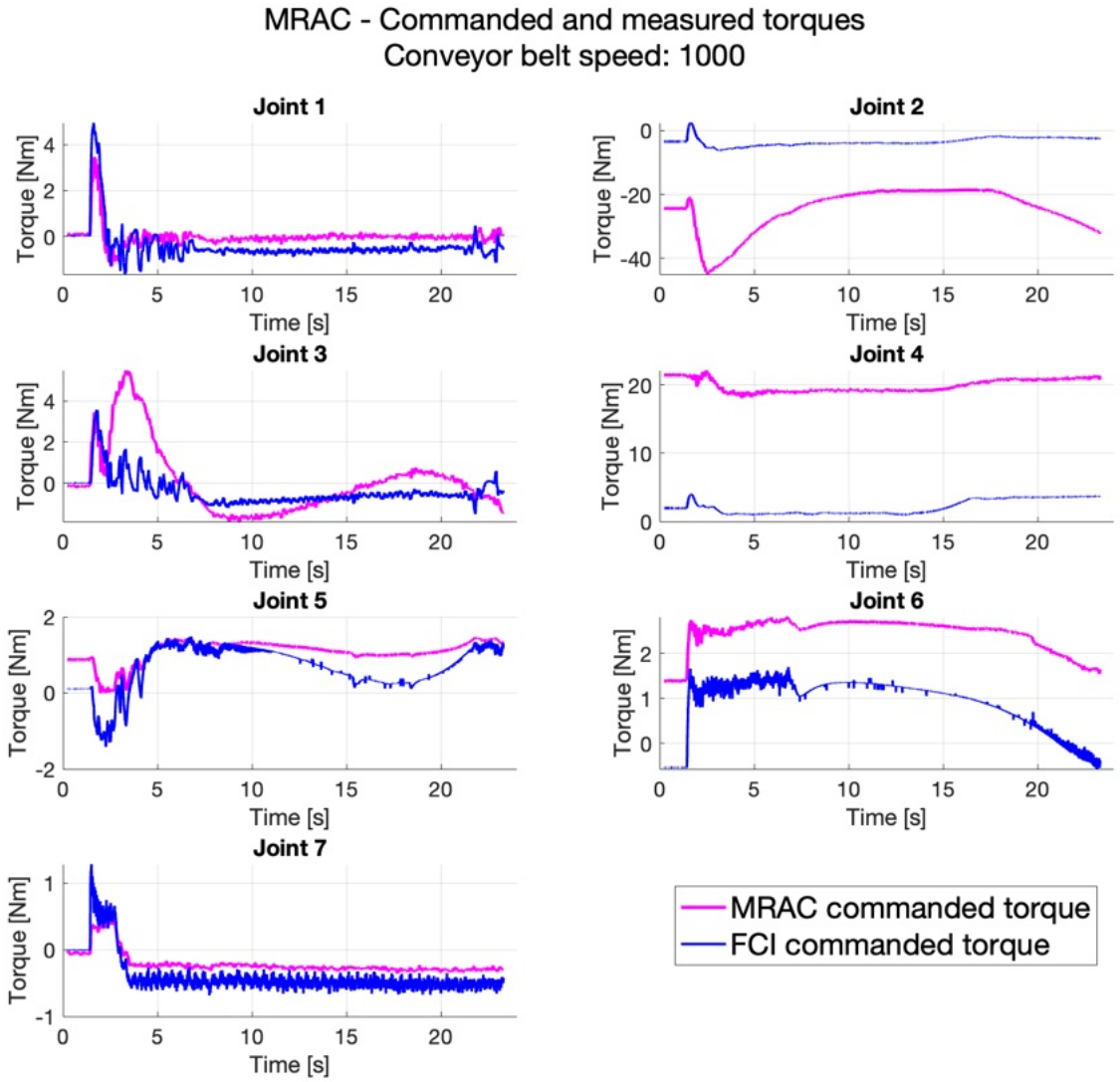


Figure E.16: Torque dynamics when tracking a conveyor belt at speed 1000. For each joint, the MRAC torque commands (in — magenta) are compared against the actual torque applied to the joint by the FCI (in — blue).

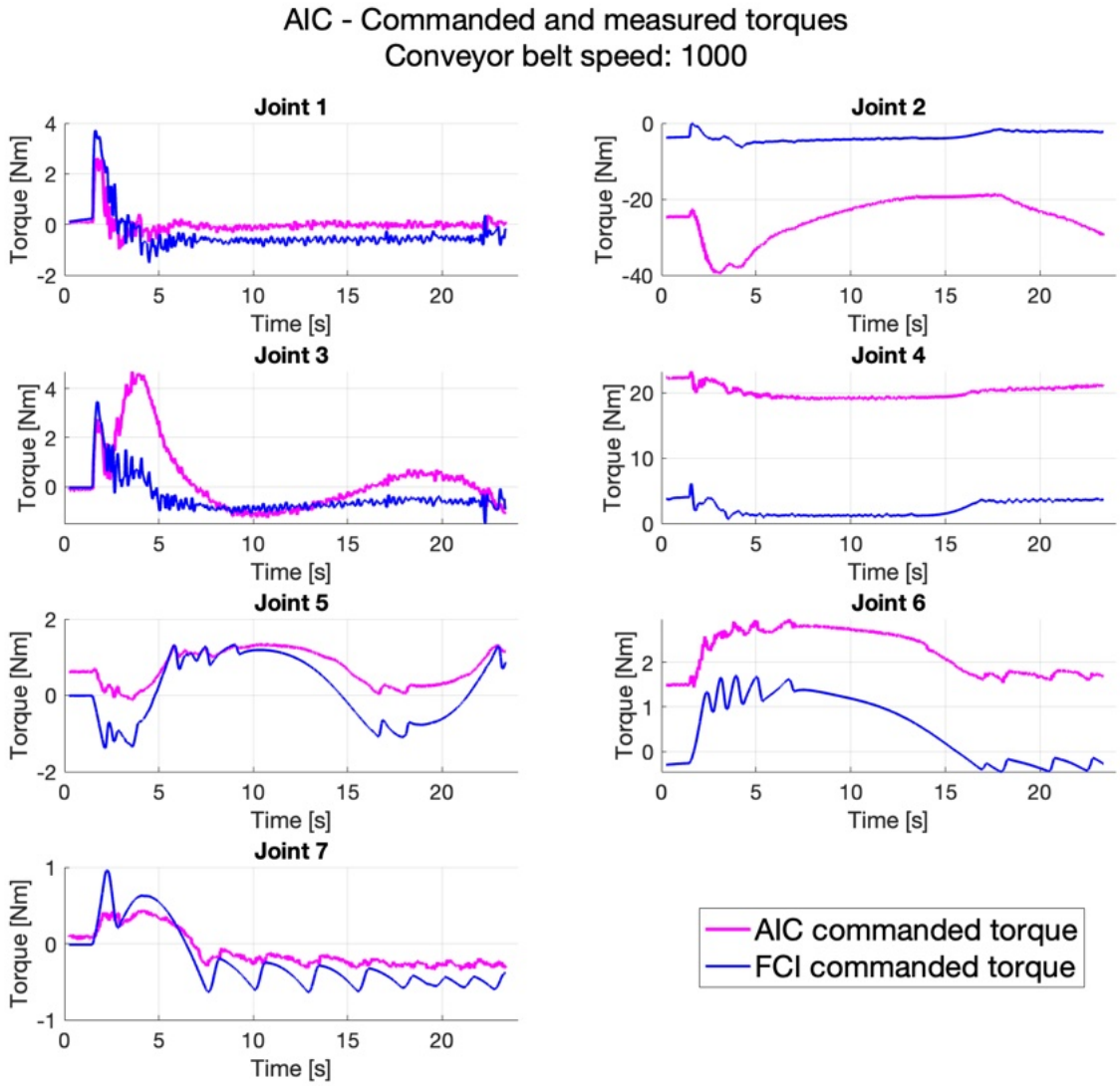


Figure E.17: Torque dynamics when tracking a conveyor belt at speed 1000. For each joint, the AIC torque commands (in — magenta) are compared against the actual torque applied to the joint by the FCI (in — blue).

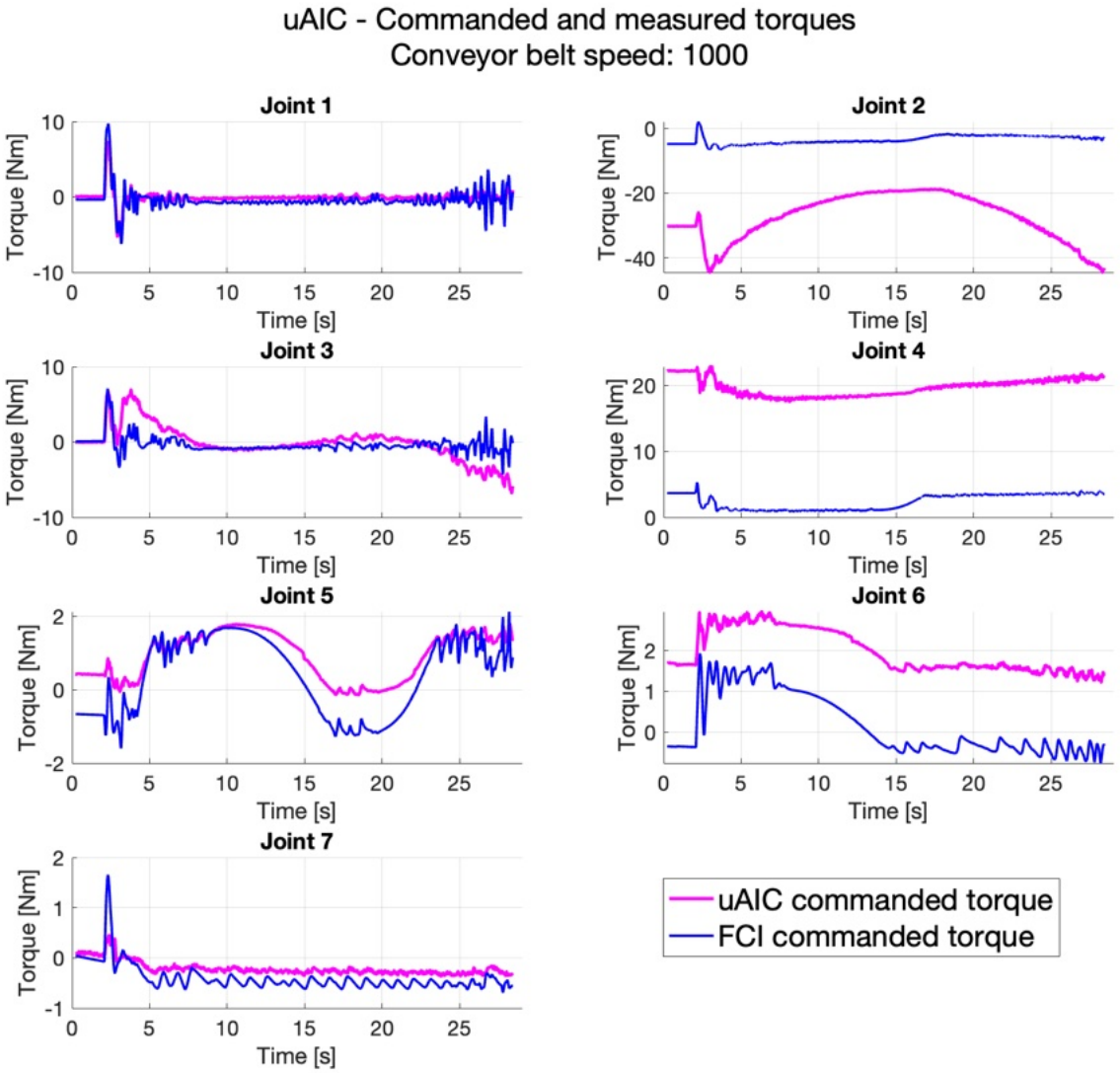


Figure E.18: Torque dynamics when tracking a conveyor belt at speed 1000. For each joint, the uAIC torque commands (in — magenta) are compared against the actual torque applied to the joint by the FCI (in — blue).

E.1.2.2 Speed 1500

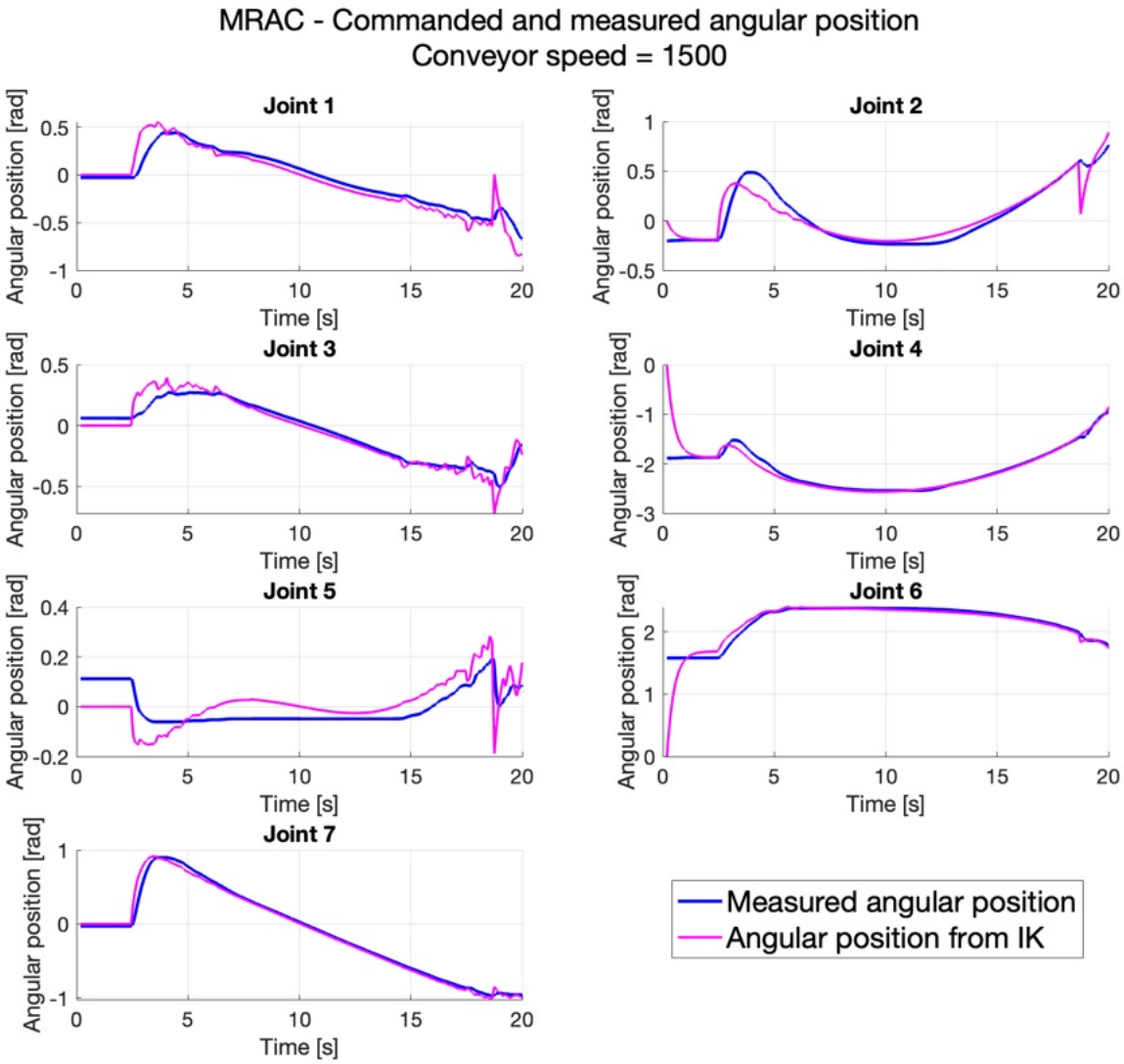


Figure E.19: Joint-space tracking of the MRAC at a conveyor belt speed of 1500. For each joint the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

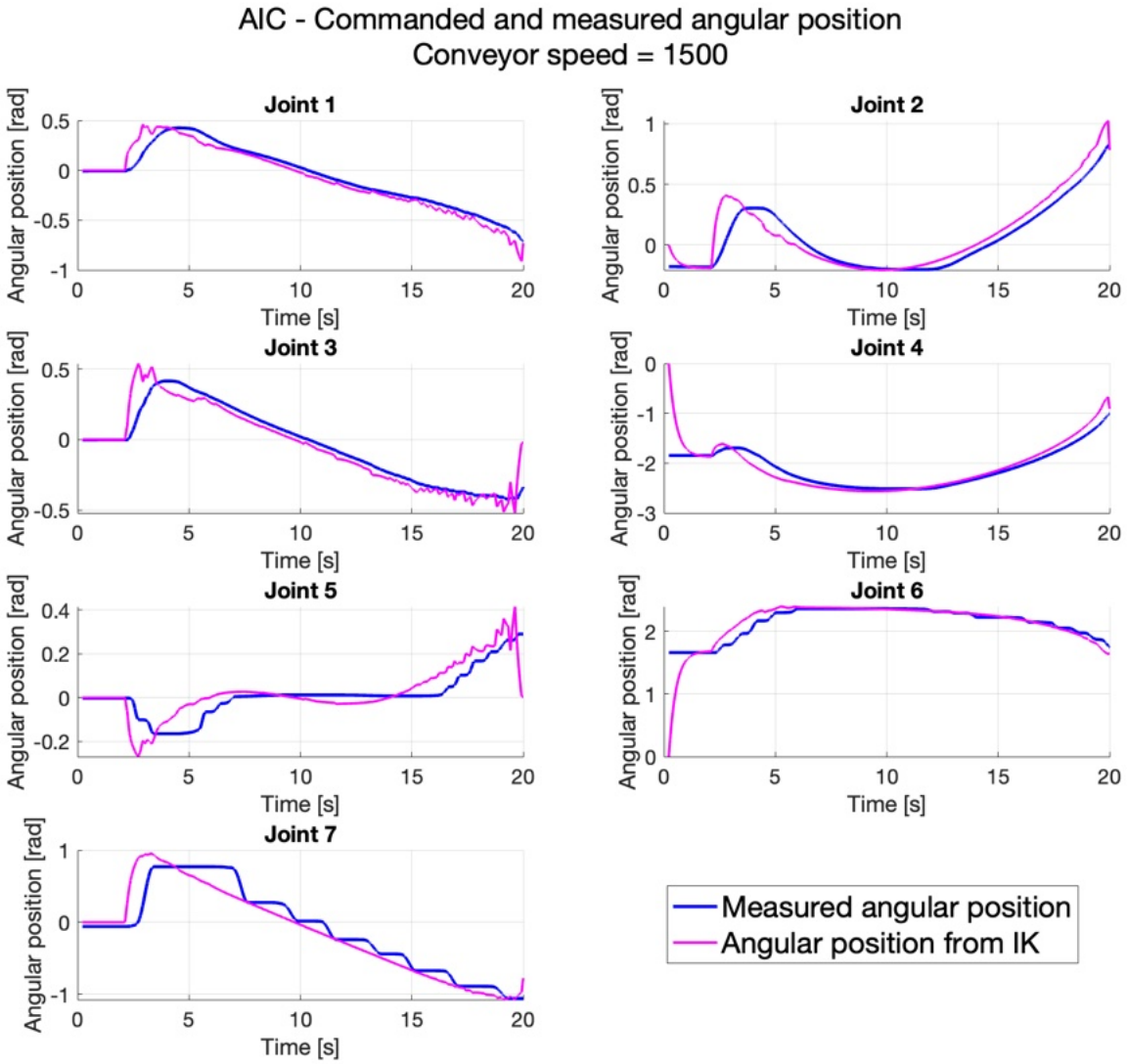


Figure E.20: Joint-space tracking of the AIC at a conveyor belt speed of 1500. For each joint the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

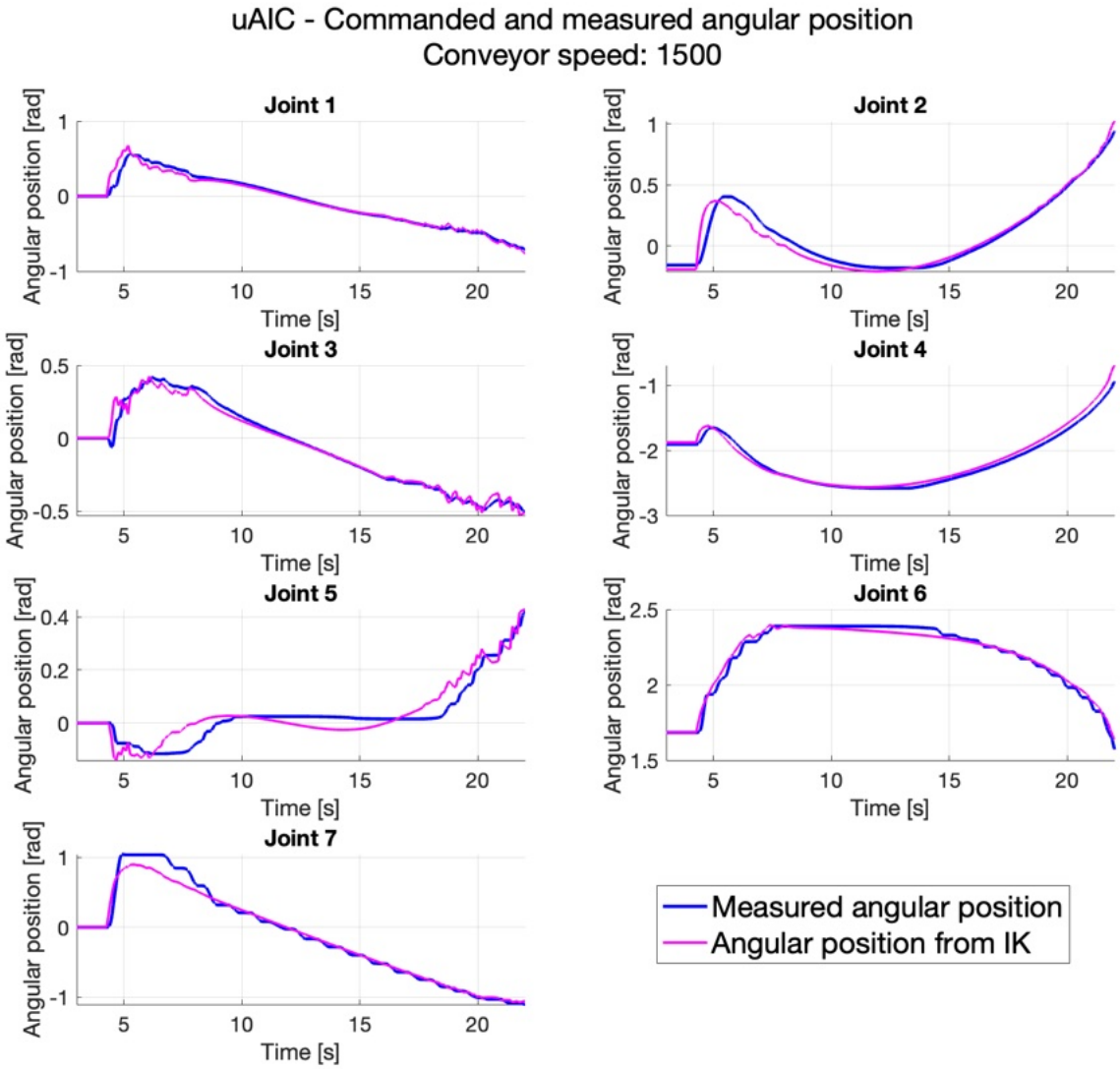


Figure E.21: Joint-space tracking of the uAIC at a conveyor belt speed of 1500. For each joint the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

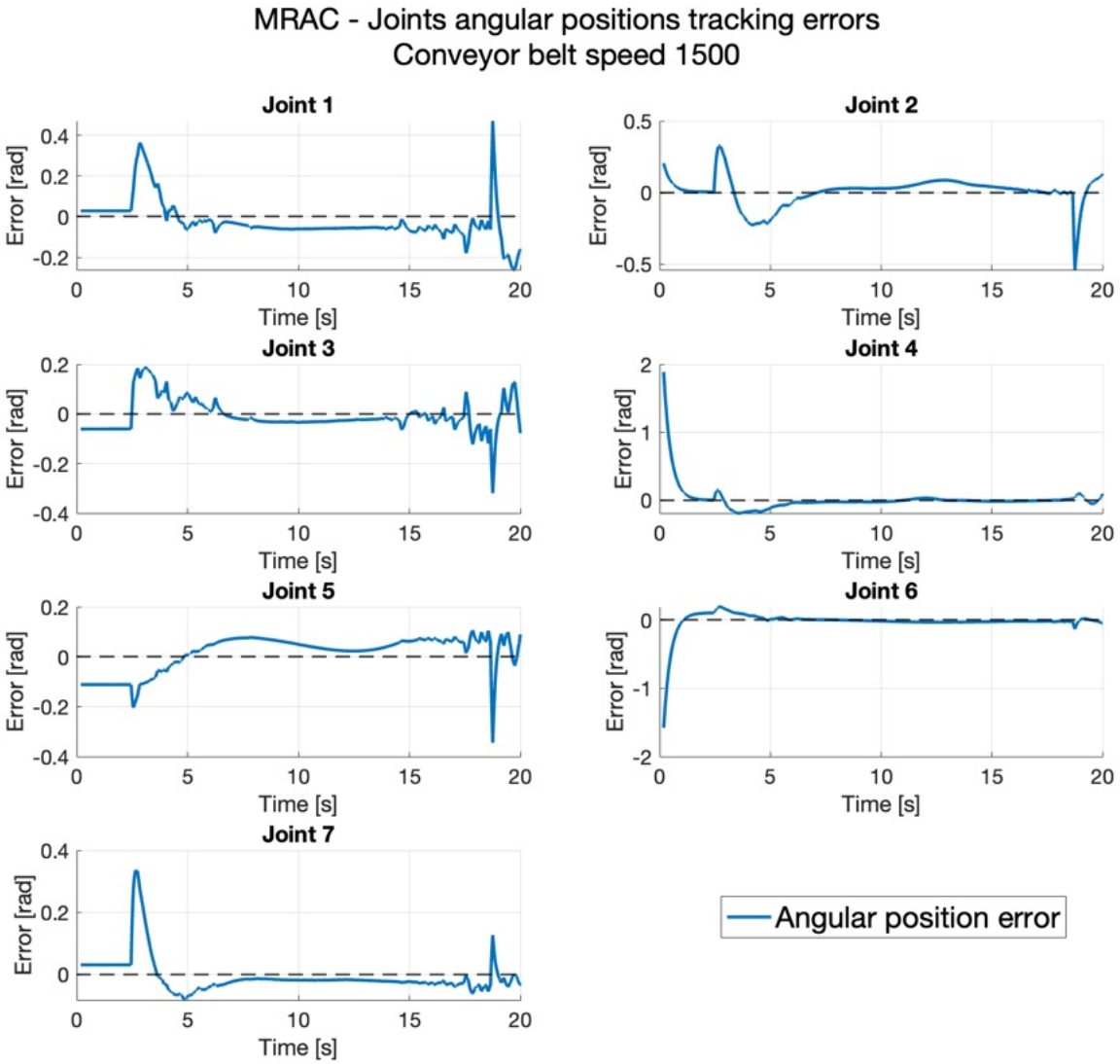


Figure E.22: Joint-space tracking errors of the MRAC at a conveyor belt speed of 1500. For each joint, the difference between the desired and measured joint angular position is shown.

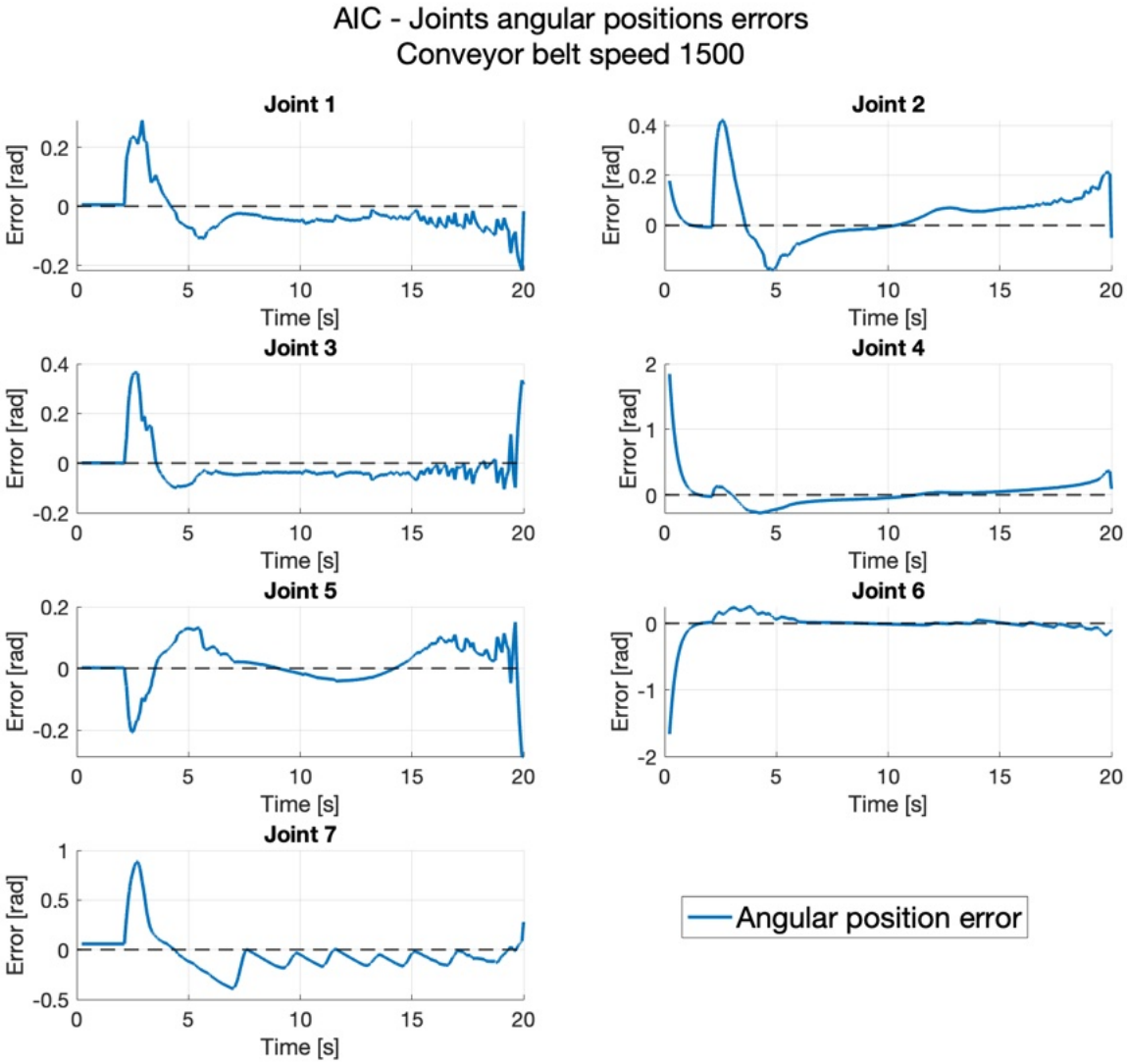


Figure E.23: Joint-space tracking errors of the AIC at a conveyor belt speed of 1500. For each joint, the difference between the desired and measured joint angular position is shown.

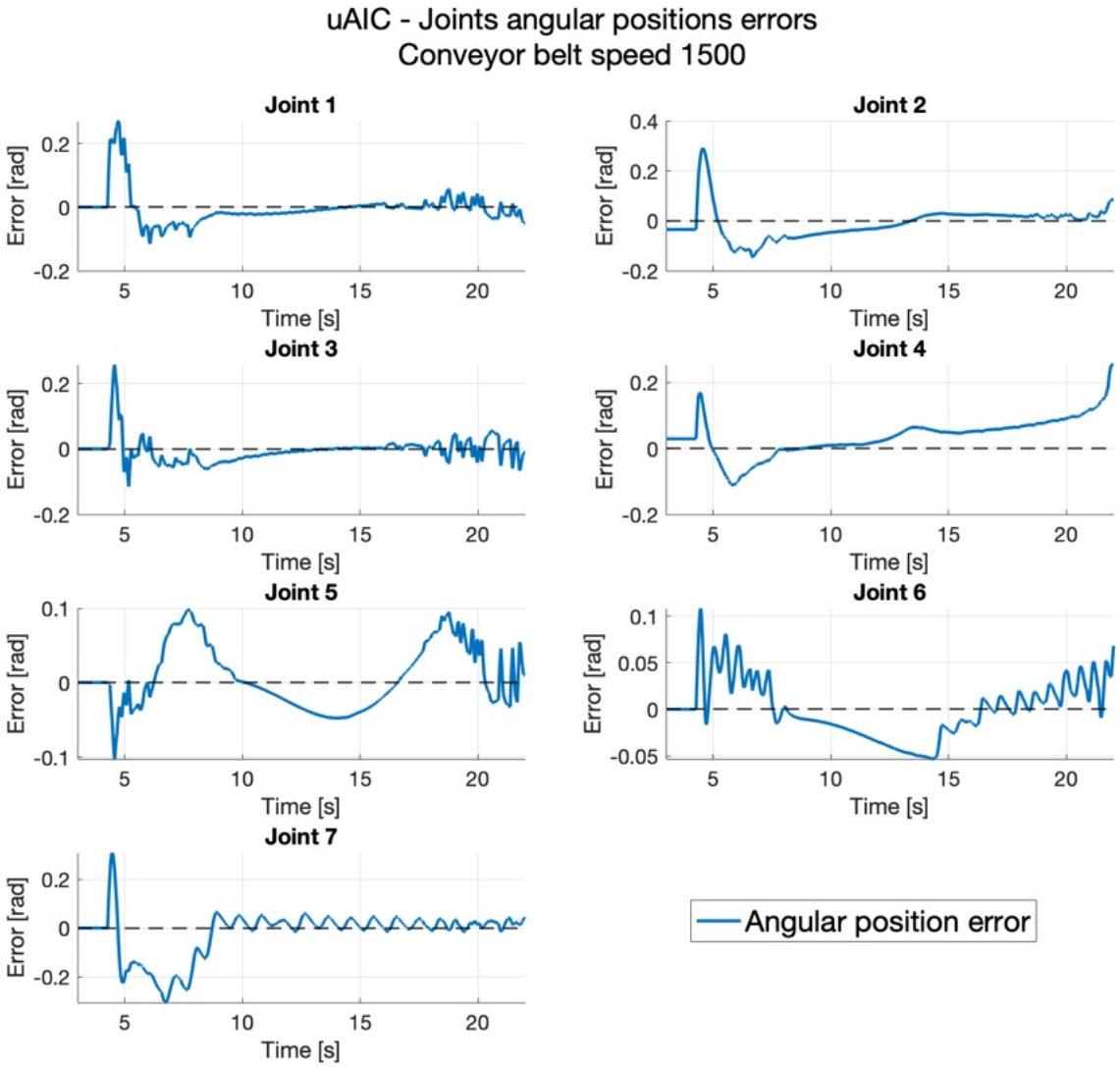


Figure E.24: Joint-space tracking errors of the uAIC at a conveyor belt speed of 1500. For each joint, the difference between the desired and measured joint angular position is shown.

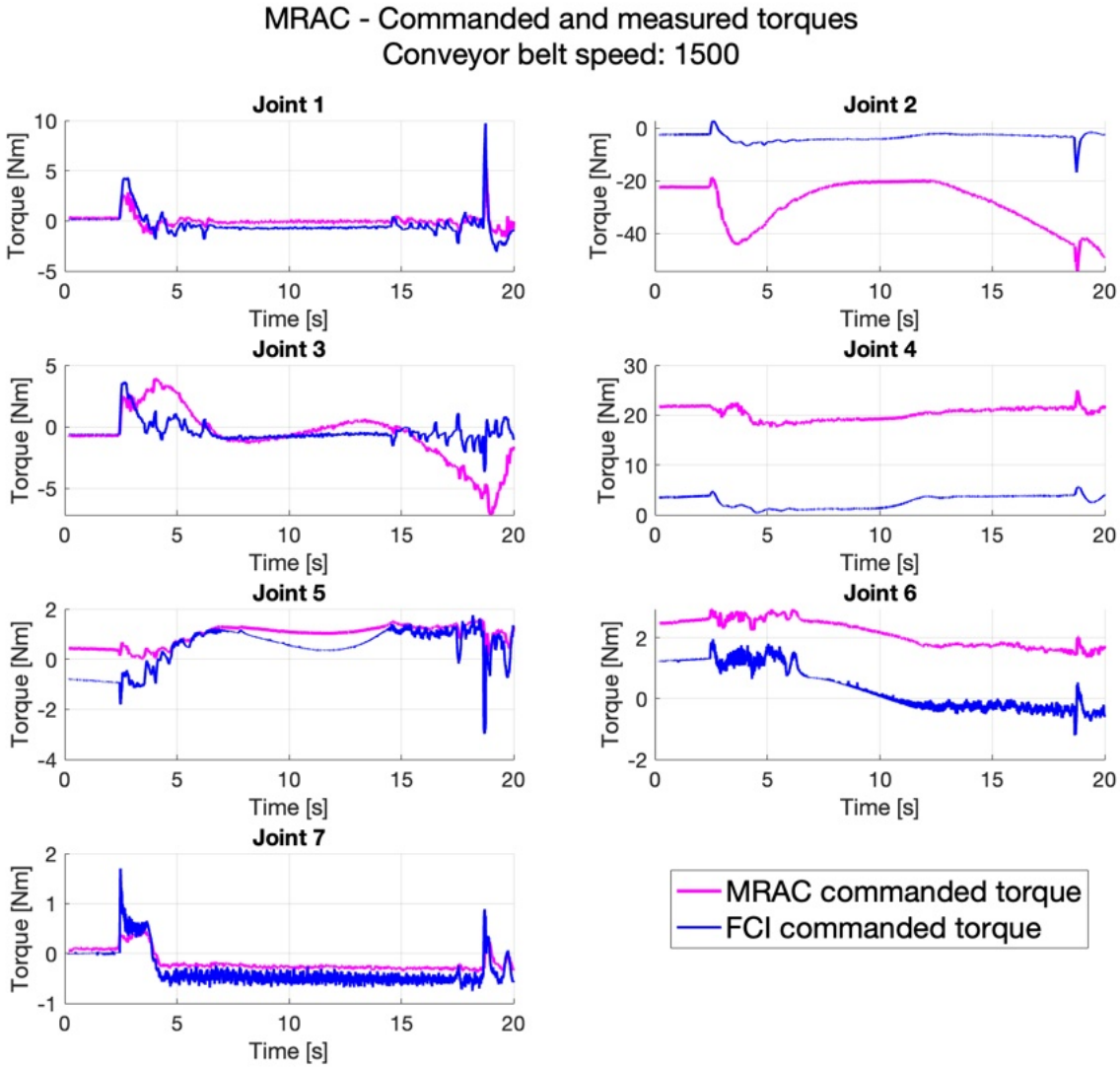


Figure E.25: Torque dynamics when tracking a conveyor belt at speed 1500. For each joint, the MRAC torque commands (in — magenta) are compared against the actual torque applied to the joint by the FCI (in — blue).

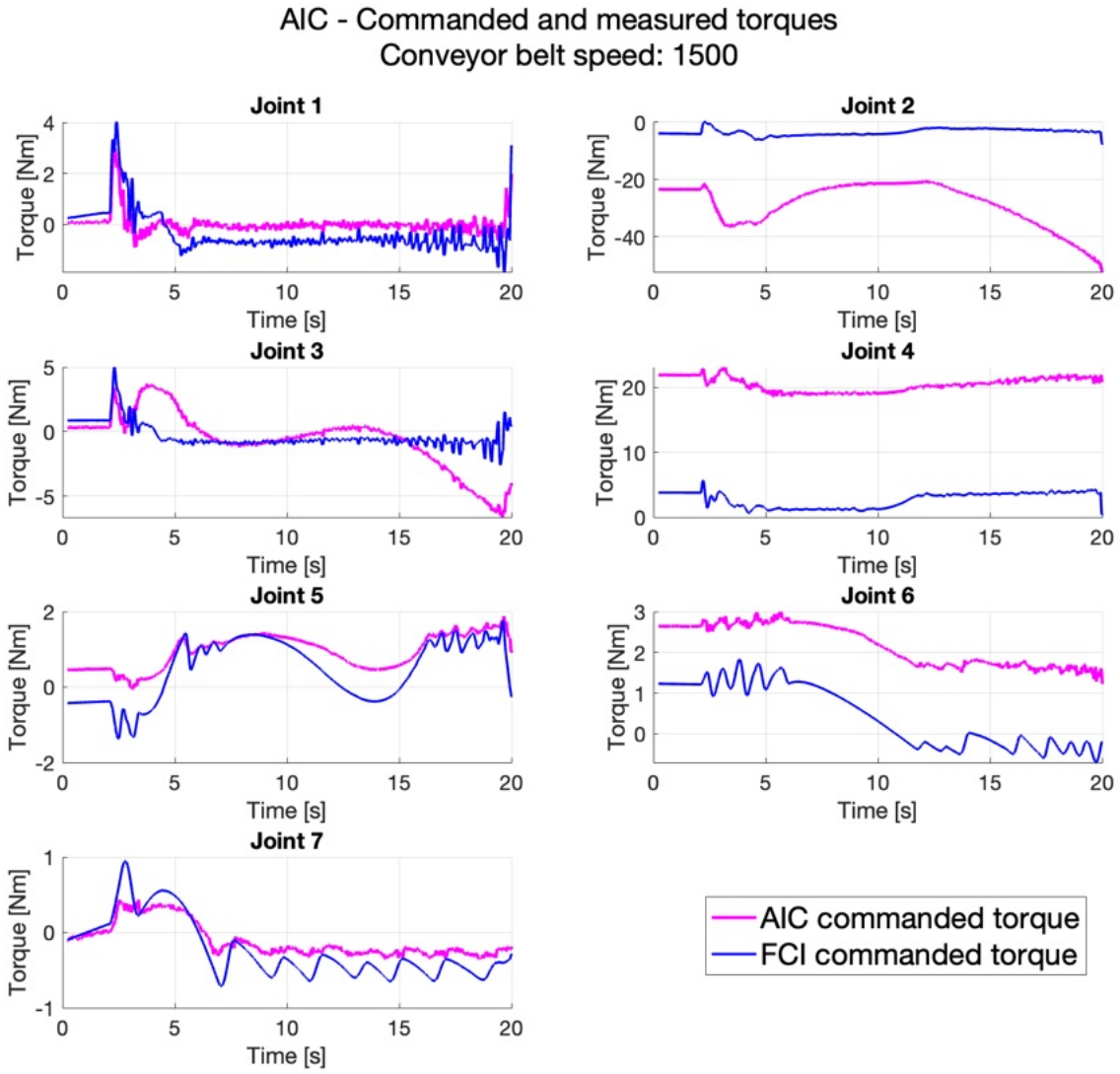


Figure E.26: Torque dynamics when tracking a conveyor belt at speed 1500. For each joint, the AIC torque commands (in — magenta) are compared against the actual torque applied to the joint by the FCI (in — blue).

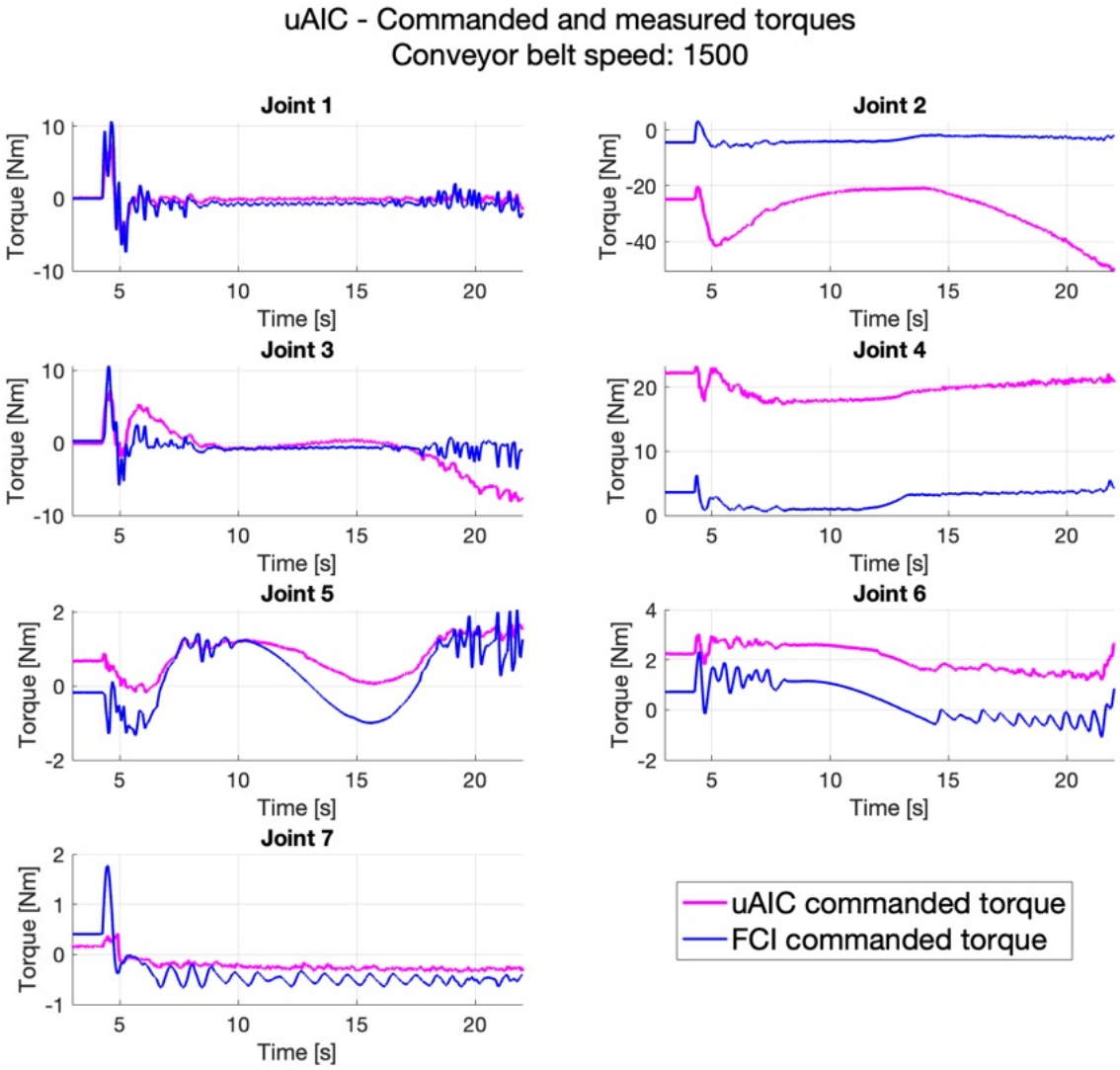


Figure E.27: Torque dynamics when tracking a conveyor belt at speed 1500. For each joint, the uAIC torque commands (in — magenta) are compared against the actual torque applied to the joint by the FCI (in — blue).

E.1.2.3 Speed 2000

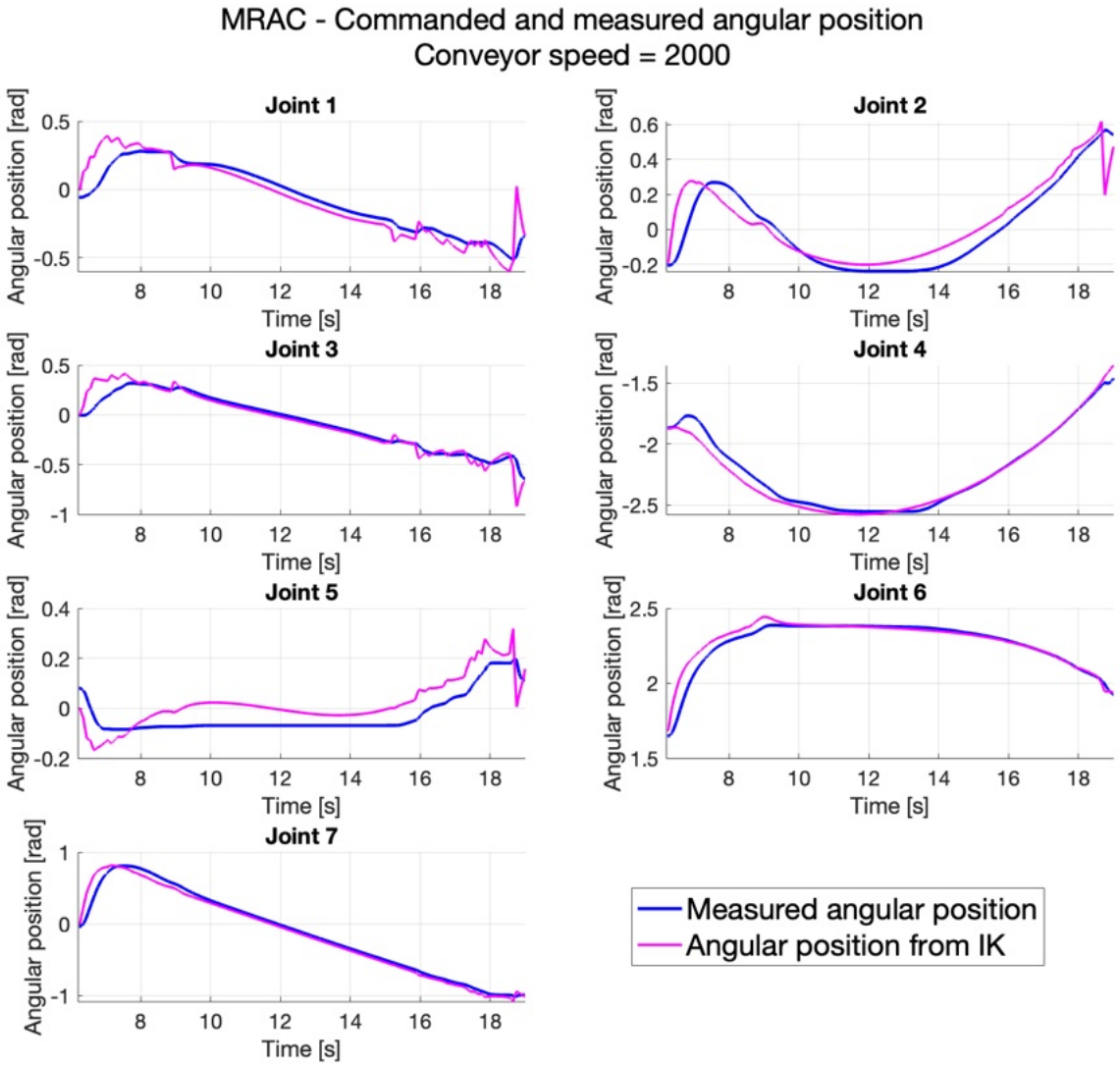


Figure E.28: Joint-space tracking of the MRAC at a conveyor belt speed of 2000. For each joint the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

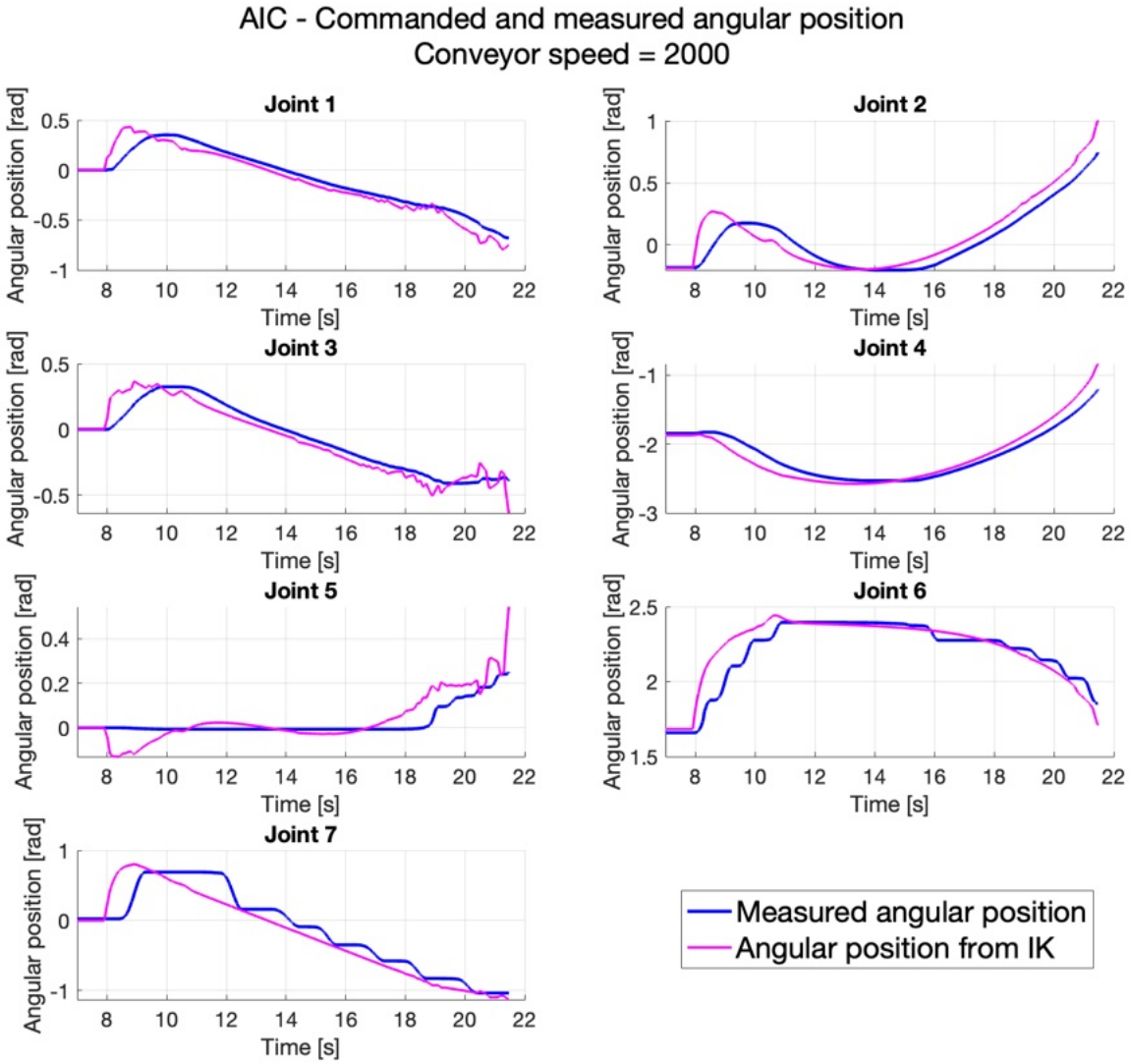


Figure E.29: Joint-space tracking of the AIC at a conveyor belt speed of 2000. For each joint the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

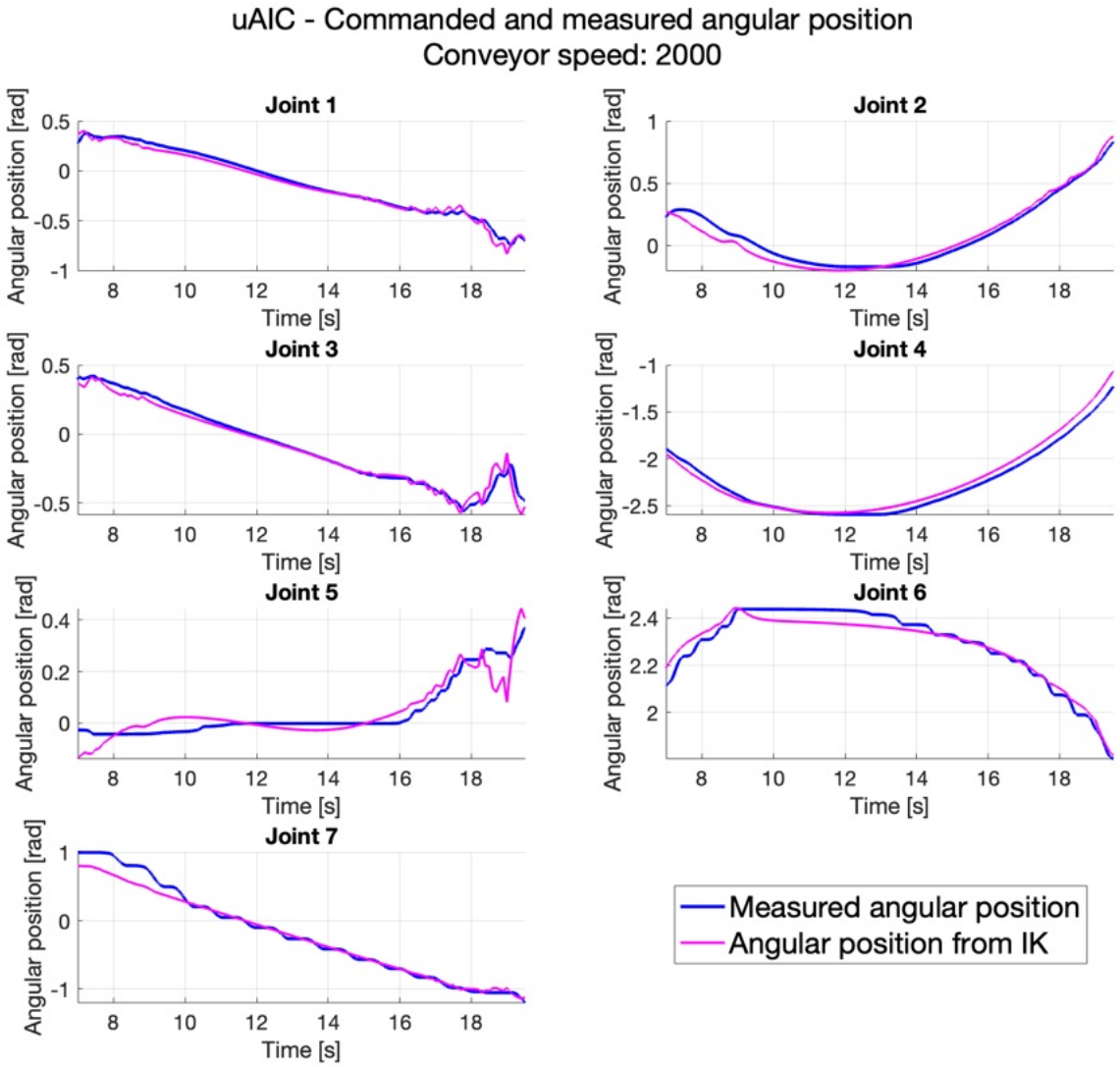


Figure E.30: Joint-space tracking of the uAIC at a conveyor belt speed of 2000. For each joint the measured joint angular position (in — blue) is compared against the desired joint angular position (in — magenta) as commanded by the Trac-IK.

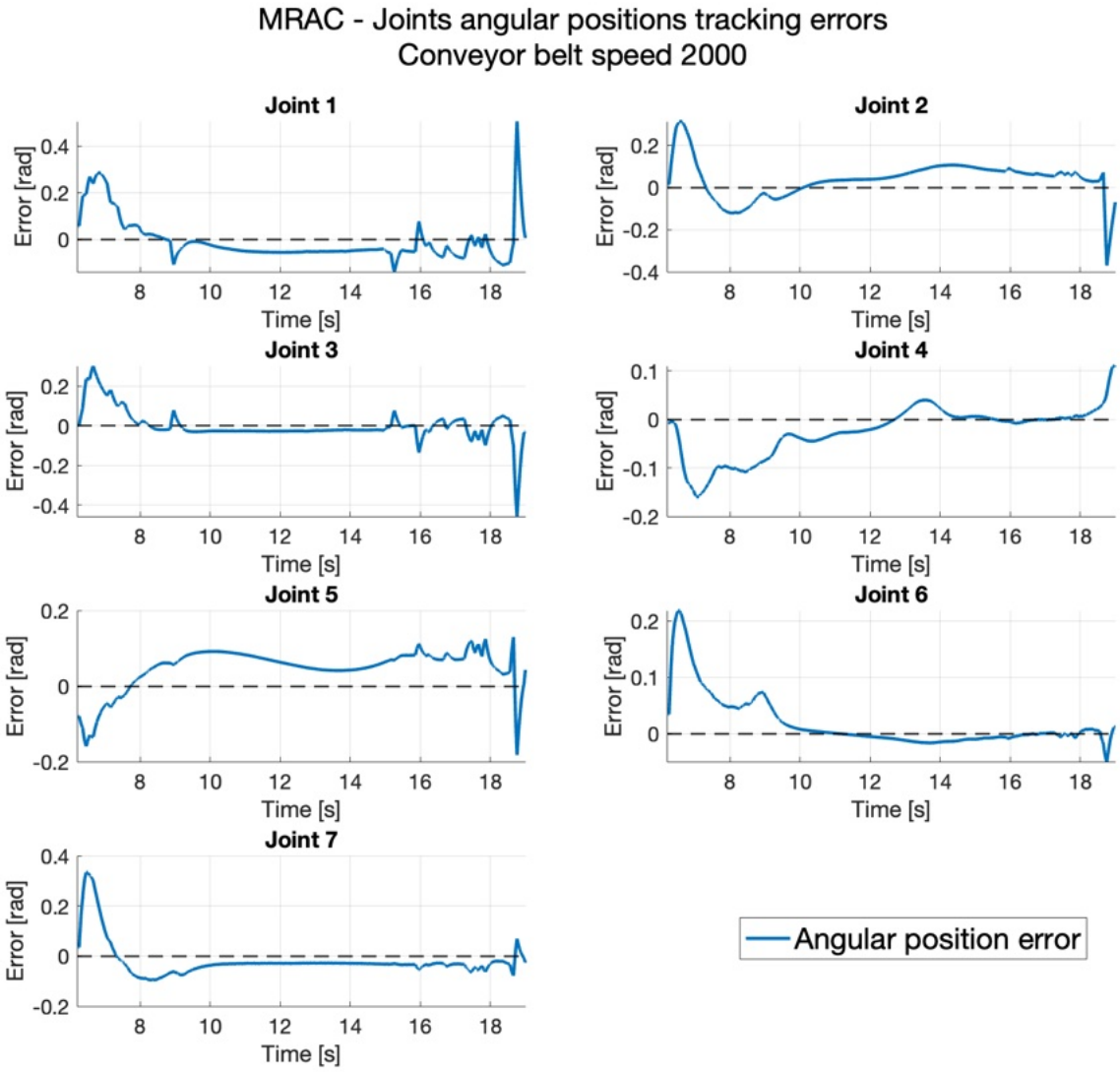


Figure E.31: Joint-space tracking error of the MRAC at a conveyor belt speed of 2000. For each joint, the difference between the desired and measured joint angular position is shown.

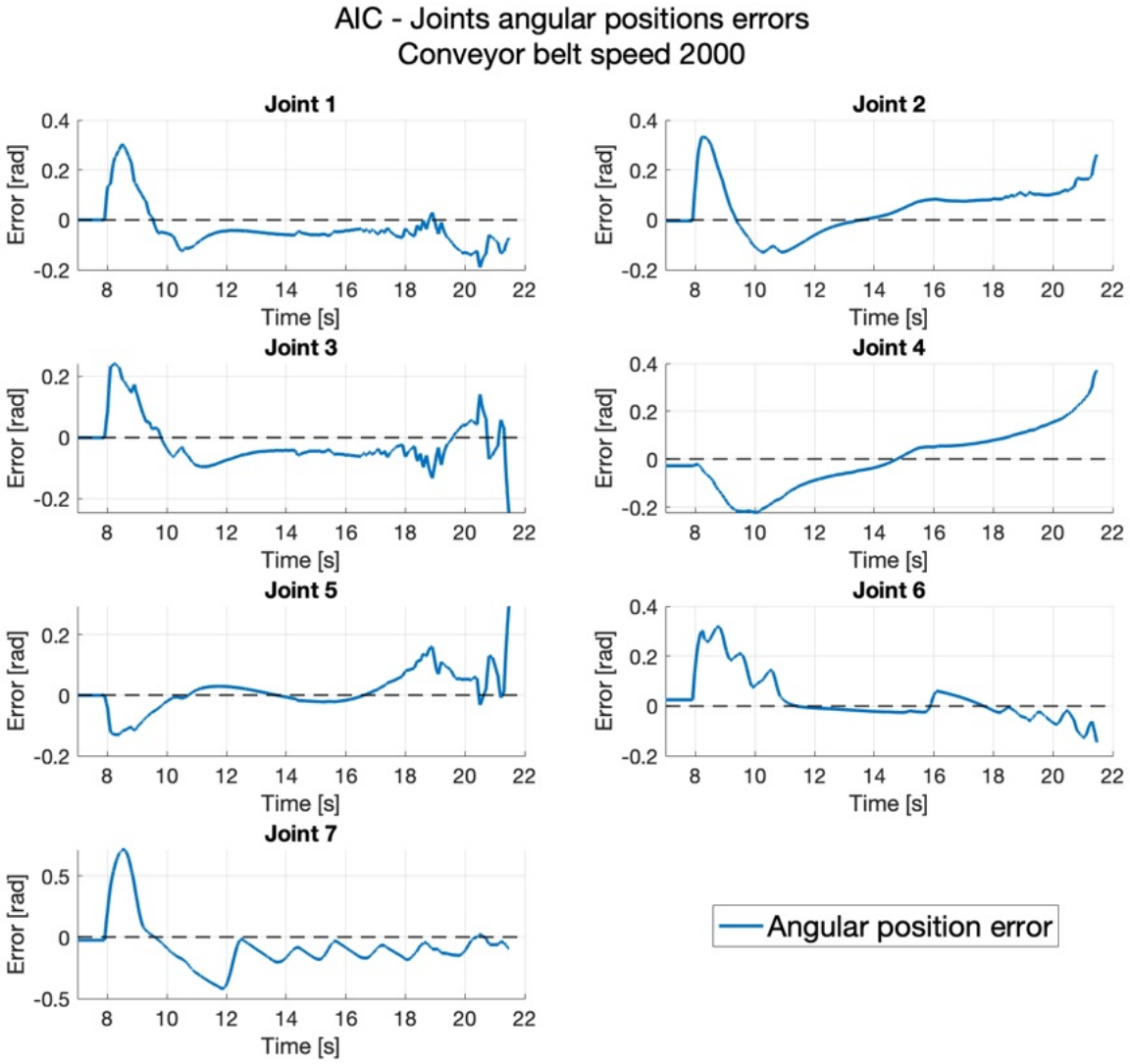


Figure E.32: Joint-space tracking error of the AIC at a conveyor belt speed of 2000. For each joint, the difference between the desired and measured joint angular position is shown.

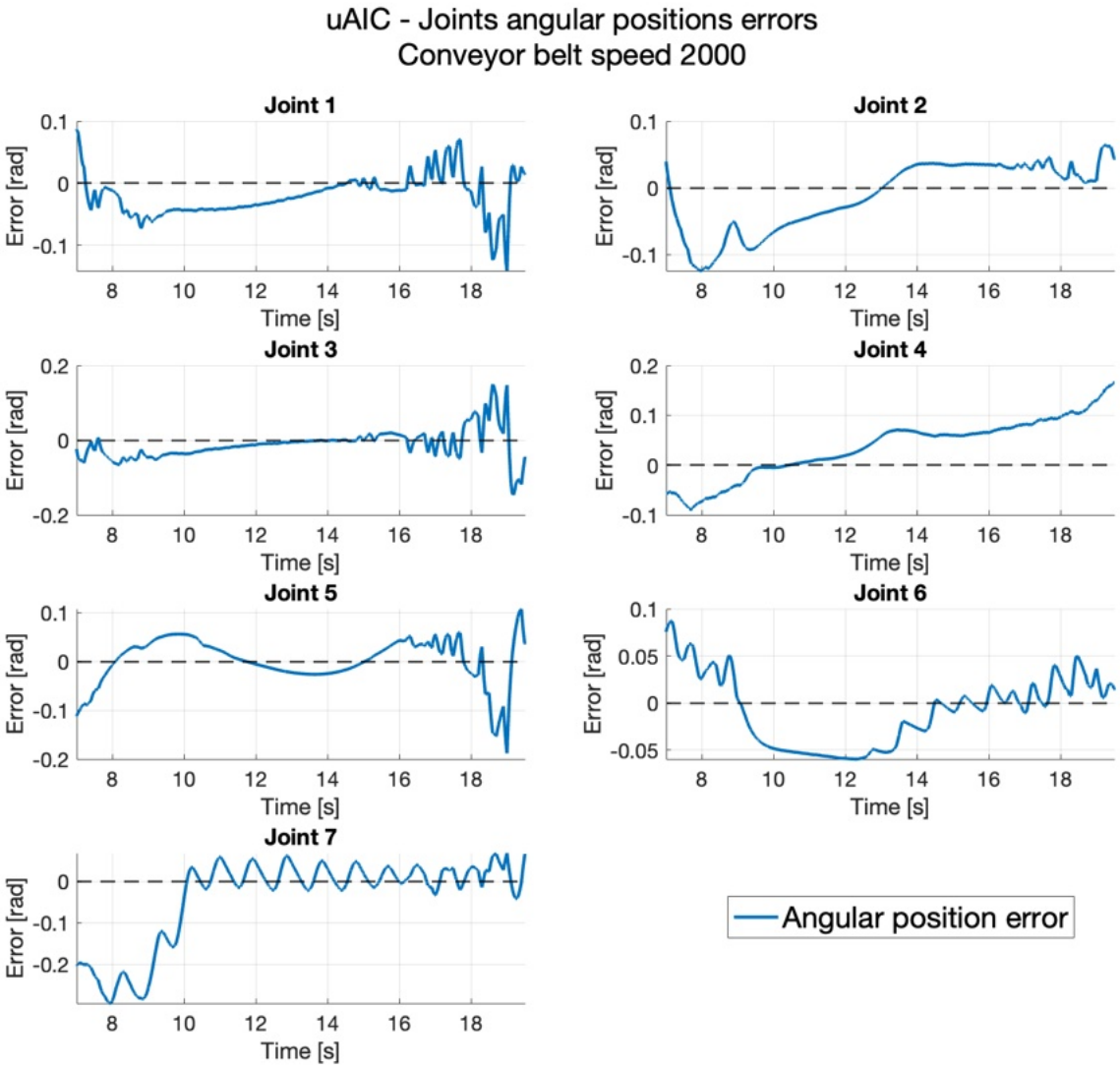


Figure E.33: Joint-space tracking error of the uAIC at a conveyor belt speed of 2000. For each joint, the difference between the desired and measured joint angular position is shown.

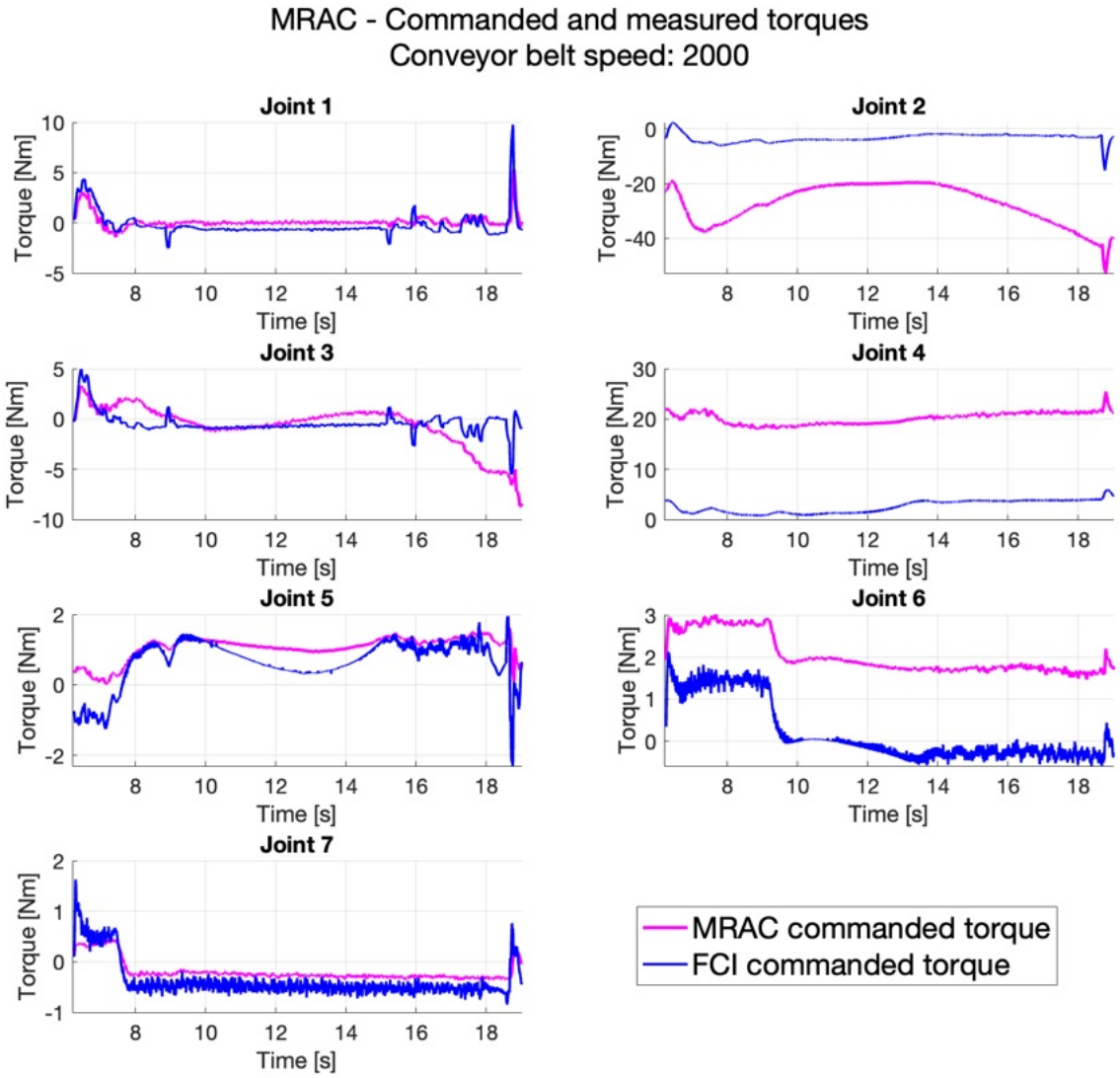


Figure E.34: Torque dynamics when tracking a conveyor belt at speed 2000. For each joint, the MRAC torque commands (in — magenta) are compared against the actual torque applied to the joint by the FCI (in — blue).

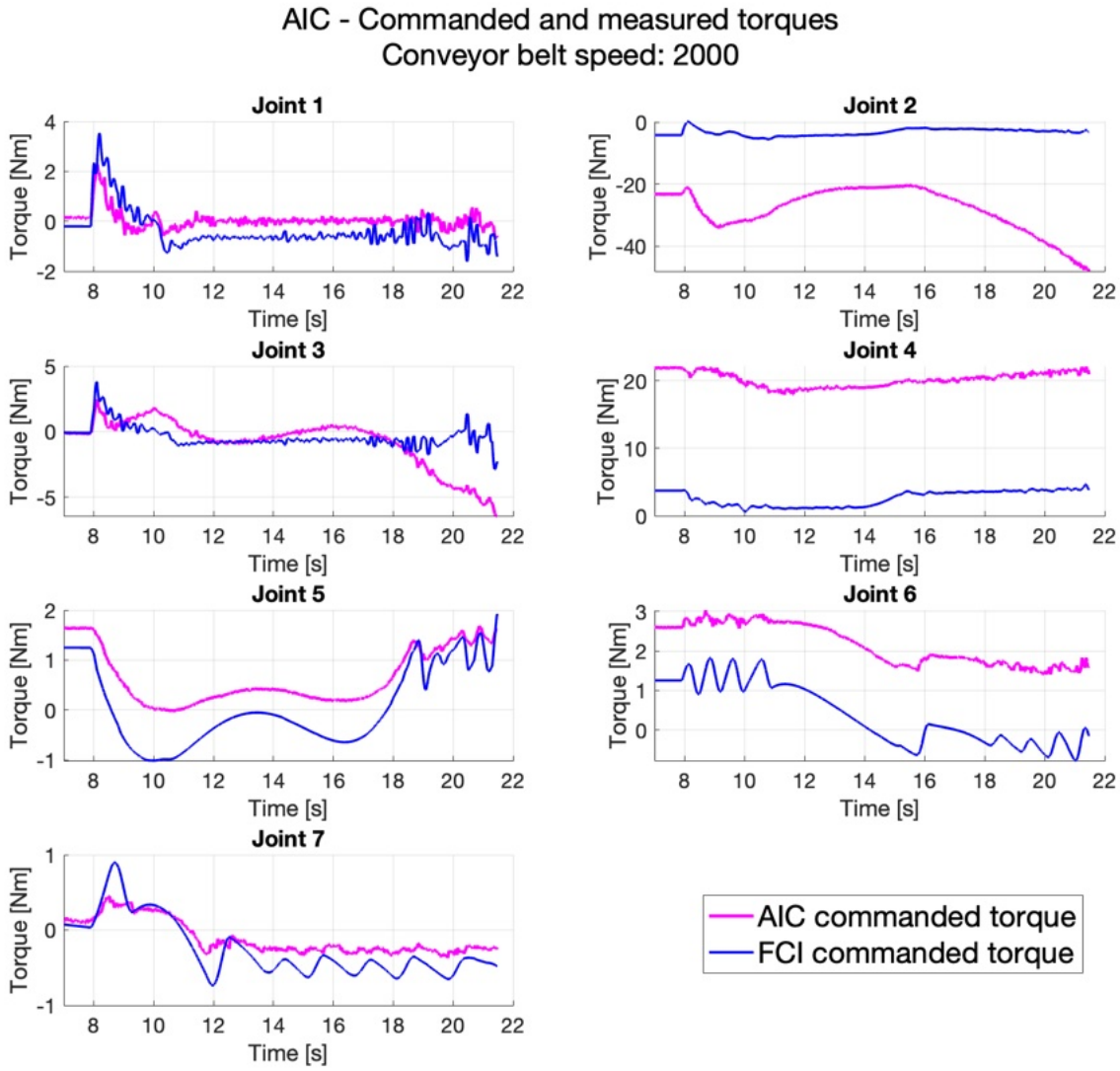


Figure E.35: Torque dynamics when tracking a conveyor belt at speed 2000. For each joint, the AIC torque commands (in — magenta) are compared against the actual torque applied to the joint by the FCI (in — blue).

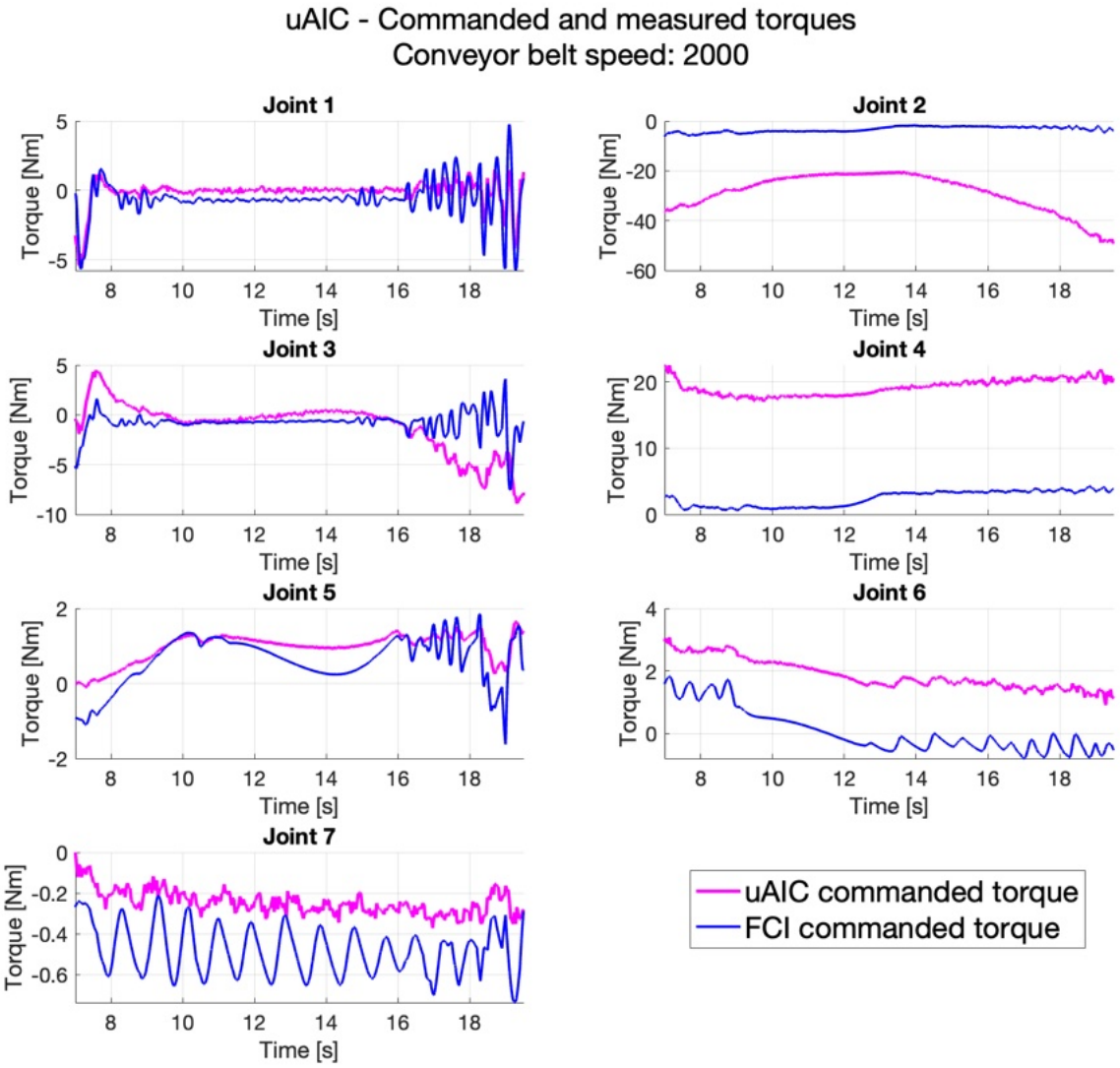


Figure E.36: Torque dynamics when tracking a conveyor belt at speed 2000. For each joint, the uAIC torque commands (in — magenta) are compared against the actual torque applied to the joint by the FCI (in — blue).

E.2 Comparison of the Performances on a Simulated E-Grocery Context

E.2.1 Grasping of Static Objects From Unknown Locations

# Test	Area	Averall Time needed	Time needed for detection of the object	Time to go to pick-position	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
1	Area 1	70,69 sec	0,2 sec	3,71 sec	0,07 sec	62,53 sec	Failure	Too slow
2	Area 1	18,44 sec	0,23 sec	2,69 sec	0,08 sec	10,9 sec	Success	
3	Area 1	30,23 sec	0,2 sec	13,87 sec	0,07 sec	11,97 sec	Success	
4	Area 1						Failure	Failed to converge to set-point
5	Area 1	17,83 sec	0,24 sec	2,77 sec	0,09 sec	10,27 sec	Success	
6	Area 1						Failure	Failed to converge to set-point
7	Area 1	71,59 sec	0,19 sec	38,09 sec	0,07 sec	10,41 sec	Failure	Too slow
8	Area 1	82,17 sec	0,2 sec	53,86 sec	0,09 sec	22,71 sec	Failure	Too slow
9	Area 1	19,58 sec	0,14 sec	4,58 sec	0,06 sec	6,91 sec	Success	
10	Area 1						Failure	Failed to converge to set-point
11	Area 1	18,53 sec	0,15 sec	5,97 sec	0,06 sec	8,03 sec	Success	
12	Area 1	20,79 sec	0,13 sec	7,72 sec	0,06 sec	8,59 sec	Success	
13	Area 1	42,64 sec	0,15 sec	6,77 sec	0,07 sec	31,29 sec	Success	
14	Area 1	68,68 sec	0,18 sec	33,2 sec	0,09 sec	30,58 sec	Failure	Too slow
15	Area 1	124,09 sec	0,19 sec	79,01 sec	0,06 sec	9,86 sec	Failure	Too slow
16	Area 2	12,93 sec	0,15 sec	2,92 sec	0,04 sec	5,45 sec	Success	
17	Area 2						Failure	Failed to converge to set-point
18	Area 2	14,02 sec	0,15 sec	3,15 sec	0,09 sec	6,43 sec	Success	
19	Area 2	21,4 sec	0,15 sec	10,17 sec	0,06 sec	6,65 sec	Success	
20	Area 2	21,41 sec	0,19 sec	10,08 sec	0,06 sec	6,81 sec	Success	
21	Area 2	24,36 sec	0,19 sec	9,35 sec	0,04 sec	7,03 sec	Success	
22	Area 2	18,45 sec	0,13 sec	8,16 sec	0,09 sec	5,69 sec	Success	
23	Area 2	25,13 sec	0,2 sec	10,34 sec	0,09 sec	7,06 sec	Success	
24	Area 2	22,23 sec	0,18 sec	8,93 sec	0,1 sec	6,21 sec	Success	
25	Area 2	23,93 sec	0,2 sec	9,48 sec	0,06 sec	6,58 sec	Success	
26	Area 2	24,11 sec	0,2 sec	9,24 sec	0,09 sec	6,78 sec	Success	
27	Area 2						Failure	Failed to converge to set-point
28	Area 2	40,98 sec	0,13 sec	28,78 sec	0,09 sec	7,34 sec	Success	
29	Area 2	23,0 sec	0,2 sec	8,16 sec	0,06 sec	7,18 sec	Success	
30	Area 2						Failure	Failed to converge to set-point
31	Area 3						Failure	Failed to converge to set-point
32	Area 3						Failure	Failed to converge to set-point
33	Area 3						Failure	Failed to converge to set-point
34	Area 3						Failure	Failed to converge to set-point
35	Area 3	74,77 sec	0,23 sec	46,13 sec	0,06 sec	6,84 sec	Failure	Too slow
36	Area 3	21,9 sec	0,15 sec	10,46 sec	0,09 sec	7,06 sec	Success	
37	Area 3	80,37 sec	0,15 sec	14,17 sec	0,04 sec	53,75 sec	Failure	Too slow
38	Area 3	89,16 sec	0,17 sec	8,09 sec	0,07 sec	36,03 sec	Failure	Too slow
39	Area 3						Failure	Failed to converge to set-point
40	Area 3	79,86 sec	0,15 sec	51,29 sec	0,06 sec	23,96 sec	Failure	Too slow
41	Area 3						Failure	Failed to converge to set-point
42	Area 3						Failure	Failed to converge to set-point
43	Area 3						Failure	Failed to converge to set-point
44	Area 3						Failure	Failed to converge to set-point
45	Area 3	116,77 sec	0,19 sec	6,68 sec	0,06 sec	49,41 sec	Failure	Too slow

Table E.1: Detailed data about the static e-grocery task conducted with the MRAC.

# Test	Area	Averall Time needed	Time needed for detection of the object	Time to go to pick-position	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
1	Area 1	14,5 sec	0,14 sec	3,46 sec	0,07 sec	6,31 sec	Success	
2	Area 1	18,68 sec	0,16 sec	7,14 sec	0,09 sec	6,74 sec	Success	
3	Area 1	14,73 sec	0,24 sec	3,75 sec	0,06 sec	6,16 sec	Success	
4	Area 1	14,73 sec	0,19 sec	3,5 sec	0,09 sec	6,33 sec	Success	
5	Area 1	14,48 sec	0,16 sec	3,48 sec	0,06 sec	6,26 sec	Success	
6	Area 1	14,98 sec	0,2 sec	3,7 sec	0,06 sec	6,32 sec	Success	
7	Area 1	14,08 sec	0,19 sec	3,29 sec	0,04 sec	6,01 sec	Success	
8	Area 1	15,79 sec	0,15 sec	4,82 sec	0,04 sec	6,22 sec	Success	
9	Area 1	15,04 sec	0,13 sec	3,85 sec	0,06 sec	6,46 sec	Success	
10	Area 1	14,73 sec	0,15 sec	3,7 sec	0,1 sec	6,22 sec	Success	
11	Area 1						Failure	Failed to converge to set-point
12	Area 1	14,73 sec	0,15 sec	3,62 sec	0,04 sec	6,27 sec	Success	
13	Area 1	14,42 sec	0,09 sec	3,35 sec	0,05 sec	6,33 sec	Success	
14	Area 1	14,46 sec	0,19 sec	3,43 sec	0,09 sec	6,34 sec	Success	
15	Area 1	14,39 sec	0,15 sec	3,38 sec	0,09 sec	6,18 sec	Success	
16	Area 2	14,56 sec	0,25 sec	4,17 sec	0,09 sec	3,45 sec	Success	
17	Area 2	14,32 sec	0,16 sec	4,09 sec	0,09 sec	5,39 sec	Success	
18	Area 2	14,46 sec	0,21 sec	4,16 sec	0,04 sec	5,55 sec	Success	
19	Area 2	14,38 sec	0,15 sec	4,07 sec	0,05 sec	5,58 sec	Success	
20	Area 2	14,36 sec	0,2 sec	4,18 sec	0,07 sec	5,38 sec	Success	
21	Area 2	14,86 sec	0,15 sec	4,05 sec	0,1 sec	5,54 sec	Success	
22	Area 2	14,49 sec	0,13 sec	4,18 sec	0,07 sec	5,49 sec	Success	
23	Area 2	14,26 sec	0,15 sec	4,09 sec	0,07 sec	5,41 sec	Success	
24	Area 2	14,62 sec	0,17 sec	4,28 sec	0,07 sec	5,42 sec	Success	
25	Area 2	14,66 sec	0,24 sec	4,13 sec	0,09 sec	6,62 sec	Success	
26	Area 2	14,28 sec	0,14 sec	4,09 sec	0,06 sec	5,37 sec	Success	
27	Area 2	14,36 sec	0,15 sec	4,14 sec	0,05 sec	5,45 sec	Success	
28	Area 2	14,35 sec	0,19 sec	4,04 sec	0,09 sec	5,53 sec	Success	
29	Area 2	14,49 sec	0,15 sec	4,12 sec	0,09 sec	5,52 sec	Success	
30	Area 2	14,36 sec	0,13 sec	4,07 sec	0,04 sec	5,54 sec	Success	
31	Area 3	15,31 sec	0,19 sec	4,0 sec	0,06 sec	6,4 sec	Success	
32	Area 3	14,58 sec	0,2 sec	4,02 sec	0,09 sec	5,72 sec	Success	
33	Area 3	14,83 sec	0,2 sec	3,86 sec	0,1 sec	6,18 sec	Success	
34	Area 3	14,58 sec	0,13 sec	4,12 sec	0,06 sec	5,56 sec	Success	
35	Area 3	14,51 sec	0,24 sec	4,06 sec	0,04 sec	5,64 sec	Success	
36	Area 3	14,53 sec	0,24 sec	4,02 sec	0,06 sec	5,69 sec	Success	
37	Area 3	15,2 sec	0,24 sec	4,09 sec	0,07 sec	6,36 sec	Success	
38	Area 3	17,16 sec	0,2 sec	5,98 sec	0,04 sec	6,29 sec	Success	
39	Area 3	16,9 sec	0,19 sec	5,89 sec	0,66 sec	6,17 sec	Success	
40	Area 3	15,02 sec	0,19 sec	3,99 sec	0,06 sec	6,35 sec	Success	
41	Area 3	17,18 sec	0,16 sec	5,96 sec	0,08 sec	6,25 sec	Success	
42	Area 3	15,01 sec	0,16 sec	3,9 sec	0,06 sec	6,33 sec	Success	
43	Area 3	14,39 sec	0,13 sec	3,81 sec	0,06 sec	5,77 sec	Success	
44	Area 3						Failure	Failed to converge to set-point
45	Area 3	17,08 sec	0,1 sec	6,0 sec	0,09 sec	6,73 sec	Success	

Table E.2: Detailed data about the static e-grocery task conducted with the AIC.

# Test	Area	Averall Time needed	Time needed for detection of the object	Time to go to pick-position	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
1	Area 1	9,76 sec	0,14 sec	1,45 sec	0,09 sec	4,48 sec	Success	
2	Area 1	9,83 sec	0,13 sec	1,62 sec	0,05 sec	4,43 sec	Success	
3	Area 1	12,58 sec	0,2 sec	1,63 sec	0,06 sec	7,08 sec	Success	
4	Area 1	9,33 sec	0,16 sec	1,52 sec	0,04 sec	3,87 sec	Success	
5	Area 1	9,39 sec	0,19 sec	1,51 sec	0,09 sec	4,07 sec	Success	
6	Area 1	9,57 sec	0,19 sec	1,59 sec	0,05 sec	4,16 sec	Success	
7	Area 1	9,97 sec	0,24 sec	1,49 sec	0,09 sec	4,59 sec	Success	
8	Area 1	9,22 sec	0,14 sec	1,6 sec	0,05 sec	3,82 sec	Success	
9	Area 1	53,93 sec	0,18 sec	1,67 sec	0,1 sec	48,43 sec	Success	
10	Area 1	9,46 sec	0,24 sec	1,58 sec	0,09 sec	3,98 sec	Success	
11	Area 1	9,33 sec	0,2 sec	1,51 sec	0,05 sec	4,08 sec	Success	
12	Area 1	25,78 sec	0,24 sec	1,7 sec	0,09 sec	20,04 sec	Success	
13	Area 1						Failure	Cartesian Relfex
14	Area 1	10,0 sec	0,24 sec	1,53 sec	0,06 sec	4,5 sec	Success	
15	Area 1	9,51 sec	0,15 sec	1,53 sec	0,09 sec	4,23 sec	Success	
16	Area 2	9,04 sec	0,17 sec	1,67 sec	0,06 sec	3,54 sec	Success	
17	Area 2	9,08 sec	0,18 sec	1,63 sec	0,06 sec	3,62 sec	Success	
18	Area 2	8,82 sec	0,16 sec	1,6 sec	0,09 sec	3,31 sec	Success	
19	Area 2	8,72 sec	0,13 sec	1,64 sec	0,06 sec	3,36 sec	Success	
20	Area 2	9,42 sec	0,17 sec	1,63 sec	0,04 sec	3,91 sec	Success	
21	Area 2	8,83 sec	0,16 sec	1,61 sec	0,07 sec	3,43 sec	Success	
22	Area 2	9,18 sec	0,19 sec	1,58 sec	0,1 sec	3,71 sec	Success	
23	Area 2	8,68 sec	0,2 sec	1,71 sec	0,06 sec	3,11 sec	Success	
24	Area 2	8,95 sec	0,18 sec	1,64 sec	0,06 sec	3,48 sec	Success	
25	Area 2	8,71 sec	0,16 sec	1,61 sec	0,06 sec	3,37 sec	Success	
26	Area 2	8,85 sec	0,2 sec	1,6 sec	0,06 sec	3,42 sec	Success	
27	Area 2	9,08 sec	0,14 sec	1,61 sec	0,06 sec	3,58 sec	Success	
28	Area 2	9,08 sec	0,24 sec	1,62 sec	0,04 sec	3,58 sec	Success	
29	Area 2	9,23 sec	0,13 sec	1,64 sec	0,06 sec	3,81 sec	Success	
30	Area 2	9,08 sec	0,2 sec	1,65 sec	0,05 sec	3,58 sec	Success	
31	Area 3	55,5 sec	0,14 sec	1,57 sec	0,09 sec	50,1 sec	Success	
32	Area 3	10,3 sec	0,2 sec	1,61 sec	0,06 sec	4,72 sec	Success	
33	Area 3	10,23 sec	0,2 sec	1,54 sec	0,08 sec	4,72 sec	Success	
34	Area 3	11,12 sec	0,2 sec	1,66 sec	0,06 sec	5,55 sec	Success	
35	Area 3	13,65 sec	0,14 sec	1,6 sec	0,07 sec	5,32 sec	Success	
36	Area 3	11,33 sec	0,2 sec	1,76 sec	0,06 sec	5,84 sec	Success	
37	Area 3	10,07 sec	0,18 sec	1,63 sec	0,06 sec	4,64 sec	Success	
38	Area 3	9,85 sec	0,18 sec	1,67 sec	0,06 sec	4,43 sec	Success	
39	Area 3	9,65 sec	0,13 sec	1,71 sec	0,06 sec	4,08 sec	Success	
40	Area 3	10,24 sec	0,19 sec	1,77 sec	0,06 sec	4,58 sec	Success	
41	Area 3	9,62 sec	0,2 sec	1,59 sec	0,09 sec	4,09 sec	Success	
42	Area 3	10,4 sec	0,17 sec	1,67 sec	0,06 sec	4,94 sec	Success	
43	Area 3	9,87 sec	0,16 sec	1,5 sec	0,08 sec	4,52 sec	Success	
44	Area 3	9,7 sec	0,2 sec	1,65 sec	0,07 sec	4,29 sec	Success	
45	Area 3	11,07 sec	0,13 sec	1,67 sec	0,06 sec	5,52 sec	Success	

Table E.3: Detailed data about the static e-grocery task conducted with the uAIC.

E.2.2 Grasping of Moving Objects From Unknown Locations

# Test	Speed	Total time	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
1	1000	3,02 sec	1,1 sec	1,92 sec	Success	
2	1000				Failure	Cartesian reflex
3	1000	3,17 sec	1,24 sec	1,93 sec	Success	
4	1000				Failure	Prediction error
5	1000	2,95 sec	1,1 sec	1,86 sec	Success	
6	1000				Failure	Prediction error
7	1000	3,2 sec	1,34 sec	1,86 sec	Success	
8	1000				Failure	Cartesian reflex
9	1000	3,26 sec	1,24 sec	2,02 sec	Success	
10	1000	2,9 sec	1,18 sec	1,72 sec	Success	
11	1000				Failure	Prediction error
12	1000				Failure	Failed to converge to set-point
13	1000	2,98 sec	1,11 sec	1,87 sec	Success	
14	1000	3,09 sec	1,25 sec	1,84 sec	Success	
15	1000				Failure	Prediction error
16	1000				Failure	Cartesian reflex
17	1000				Failure	Failed to converge to set-point
18	1000	3,11 sec	1,25 sec	1,86 sec	Success	
19	1000				Failure	Failed to converge to set-point
20	1000	3,23 sec	1,24 sec	1,99 sec	Success	
21	1000				Failure	Prediction error
22	1000				Failure	Prediction error
23	1000				Failure	Prediction error
24	1000	2,73 sec	1,06 sec	1,67 sec	Success	
25	1000				Failure	Prediction error
26	1000	3,38 sec	1,0 sec	2,38 sec	Success	
27	1000				Failure	Prediction error
28	1000				Failure	Failed to converge to set-point
29	1000	2,54 sec	0,91 sec	1,62 sec	Success	
30	1000				Failure	Prediction error
31	1500	2,68 sec	0,97 sec	1,71 sec	Success	
32	1500				Failure	Failed to converge to set-point
33	1500				Failure	Prediction error
34	1500	3,06 sec	1,14 sec	1,91 sec	Success	
35	1500				Failure	Failed to converge to set-point
36	1500	2,67 sec	1,0 sec	1,67 sec	Success	
37	1500	2,65 sec	0,97 sec	1,68 sec	Success	
38	1500	2,49 sec	0,95 sec	1,54 sec	Success	
39	1500				Failure	Failed to converge to set-point
40	1500				Failure	Failed to converge to set-point
41	1500	2,98 sec	1,08 sec	1,91 sec	Success	
42	1500				Failure	Failed to converge to set-point
43	1500	3,07 sec	1,11 sec	1,96 sec	Success	
44	1500	3,18 sec	1,2 sec	1,98 sec	Success	
45	1500	3,06 sec	1,15 sec	1,92 sec	Success	

Table E.4: Detailed data about the dynamic e-grocery task conducted with the MRAC and including prediction errors - Part 1 of 2

# Test	Speed	Total time	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
46	1500	3,44 sec	1,21 sec	2,23 sec	Success	
47	1500				Failure	Prediction error
48	1500	3,19 sec	1,15 sec	2,04 sec	Success	
49	1500	3,16 sec	1,25 sec	1,91 sec	Success	
50	1500				Failure	Prediction error
51	1500	3,15 sec	1,37 sec	1,78 sec	Success	
52	1500	3,26 sec	1,24 sec	2,02 sec	Success	
53	1500				Failure	Failed to converge to set-point
54	1500	3,11 sec	1,22 sec	1,89 sec	Success	
55	1500				Failure	Failed to converge to set-point
56	1500	3,02 sec	1,09 sec	1,94 sec	Success	
57	1500	3,3 sec	1,31 sec	2,0 sec	Success	
58	1500	2,78 sec	1,04 sec	1,74 sec	Success	
59	1500	3,04 sec	1,16 sec	1,88 sec	Success	
60	1500				Failure	Failed to converge to set-point
61	2000	3,66 sec	1,11 sec	2,55 sec	Success	
62	2000	2,75 sec	1,0 sec	1,75 sec	Success	
63	2000	3,66 sec	1,11 sec	2,55 sec	Success	
64	2000	3,66 sec	1,11 sec	2,55 sec	Success	
65	2000				Failure	Failed to converge to set-point
66	2000	3,66 sec	1,11 sec	2,55 sec	Success	
67	2000	2,75 sec	1,0 sec	1,75 sec	Success	
68	2000	2,84 sec	1,01 sec	1,84 sec	Success	
69	2000				Failure	Failed to converge to set-point
70	2000	3,04 sec	1,16 sec	1,88 sec	Success	
71	2000	2,9 sec	1,01 sec	1,89 sec	Success	
72	2000				Failure	Failed to converge to set-point
73	2000	2,91 sec	1,08 sec	1,83 sec	Success	
74	2000	2,98 sec	1,09 sec	1,89 sec	Success	
75	2000	3,01 sec	1,12 sec	1,89 sec	Success	
76	2000	2,95 sec	1,12 sec	1,82 sec	Success	
77	2000				Failure	Failed to converge to set-point
78	2000				Failure	Failed to converge to set-point
79	2000				Failure	Failed to converge to set-point
80	2000				Failure	Prediction error
81	2000	3,03 sec	1,19 sec	1,84 sec	Success	
82	2000				Failure	Failed to converge to set-point
83	2000				Failure	Failed to converge to set-point
84	2000	2,94 sec	1,06 sec	1,88 sec	Success	
85	2000				Failure	Prediction error
86	2000	3,1 sec	1,11 sec	1,99 sec	Success	
87	2000				Failure	Prediction error
88	2000				Failure	Prediction error
89	2000				Failure	Prediction error
90	2000	12460,0 sec	1048,0 sec	11412,0 sec	Success	

Table E.5: Detailed data about the dynamic e-grocery task conducted with the MRAC and including prediction errors - Part 2 of 2

# Test	Speed	Total time	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
1	1000				Failure	Failed to converge to set-point
2	1000	5,03 sec	1,57 sec	3,46 sec	Success	
3	1000	5,02 sec	1,56 sec	3,45 sec	Success	
4	1000				Failure	Prediction error
5	1000	4,92 sec	1,47 sec	3,45 sec	Success	
6	1000	5,09 sec	1,59 sec	3,5 sec	Success	
7	1000	4,86 sec	1,52 sec	3,34 sec	Success	
8	1000	4,77 sec	1,4 sec	3,37 sec	Success	
9	1000				Failure	Prediction error
10	1000	4,93 sec	1,54 sec	3,39 sec	Success	
11	1000	5,06 sec	1,58 sec	3,48 sec	Success	
12	1000				Failure	Cartesian reflex
13	1000				Failure	Failed to converge to set-point
14	1000	4,96 sec	1,56 sec	3,39 sec	Success	
15	1000	6,01 sec	1,59 sec	4,42 sec	Success	
16	1000	5,04 sec	1,6 sec	3,44 sec	Success	
17	1000				Failure	Prediction error
18	1000				Failure	Cartesian reflex
19	1000				Failure	Prediction error
20	1000				Failure	Prediction error
21	1000	7,12 sec	3,41 sec	3,71 sec	Success	
22	1000	4,95 sec	1,66 sec	3,3 sec	Success	
23	1000				Failure	Prediction error
24	1000				Failure	Prediction error
25	1000				Failure	Cartesian reflex
26	1000				Failure	Prediction error
27	1000				Failure	Prediction error
28	1000	5,18 sec	1,76 sec	3,42 sec	Success	
29	1000				Failure	Failed to converge to set-point
30	1000	4,92 sec	1,59 sec	3,34 sec	Success	
31	1500	5,61 sec	2,23 sec	3,38 sec	Success	
32	1500				Failure	Prediction error
33	1500				Failure	Prediction error
34	1500	7,22 sec	3,66 sec	3,56 sec	Success	
35	1500				Failure	Failed to converge to set-point
36	1500	6,4 sec	2,95 sec	3,46 sec	Success	
37	1500	5,87 sec	2,45 sec	3,43 sec	Success	
38	1500				Failure	Prediction error
39	1500	4,96 sec	1,56 sec	3,39 sec	Success	
40	1500				Failure	Prediction error
41	1500				Failure	Failed to converge to set-point
42	1500	6,57 sec	2,95 sec	3,62 sec	Success	
43	1500				Failure	Prediction error
44	1500				Failure	Failed to converge to set-point
45	1500	6,18 sec	2,68 sec	3,49 sec	Success	

Table E.6: Detailed data about the dynamic e-grocery task conducted with the AIC and including prediction errors - Part 1 of 2

# Test	Speed	Total time	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
46	1500	6,39 sec	3,05 sec	3,34 sec	Success	
47	1500				Failure	Failed to converge to set-point
48	1500	5,49 sec	2,16 sec	3,33 sec	Success	
49	1500	4,91 sec	1,52 sec	3,39 sec	Success	
50	1500	4,77 sec	1,38 sec	3,39 sec	Success	
51	1500				Failure	Prediction error
52	1500	6,19 sec	2,66 sec	3,53 sec	Success	
53	1500	5,71 sec	2,2 sec	3,51 sec	Success	
54	1500				Failure	Prediction error
55	1500	5,98 sec	2,45 sec	3,53 sec	Success	
56	1500	5,31 sec	1,76 sec	3,56 sec	Success	
57	1500	5,16 sec	1,58 sec	3,58 sec	Success	
58	1500				Failure	Prediction error
59	1500				Failure	Prediction error
60	1500				Failure	Failed to converge to set-point
61	2000				Failure	Prediction error
62	2000	8,11 sec	4,65 sec	3,46 sec	Success	
63	2000	6,03 sec	2,66 sec	3,36 sec	Success	
64	2000	6,45 sec	3,07 sec	3,38 sec	Success	
65	2000				Failure	Failed to converge to set-point
66	2000	6,12 sec	2,68 sec	3,43 sec	Success	
67	2000				Failure	Prediction error
68	2000				Failure	Prediction error
69	2000				Failure	Prediction error
70	2000	7,65 sec	4,24 sec	3,41 sec	Success	
71	2000				Failure	Failed to converge to set-point
72	2000				Failure	Prediction error
73	2000				Failure	Failed to converge to set-point
74	2000	5,58 sec	2,14 sec	3,44 sec	Success	
75	2000	7,4 sec	3,88 sec	3,52 sec	Success	
76	2000				Failure	Prediction error
77	2000	6,1 sec	2,79 sec	3,31 sec	Success	
78	2000				Failure	Prediction error
79	2000				Failure	Prediction error
80	2000	7,26 sec	3,85 sec	3,42 sec	Success	
81	2000	7,46 sec	3,97 sec	3,49 sec	Success	
82	2000	7,07 sec	3,56 sec	3,51 sec	Success	
83	2000				Failure	Failed to converge to set-point
84	2000				Failure	Failed to converge to set-point
85	2000				Failure	Failed to converge to set-point
86	2000				Failure	Failed to converge to set-point
87	2000	7,17 sec	3,7 sec	3,47 sec	Success	
88	2000				Failure	Failed to converge to set-point
89	2000				Failure	Prediction error
90	2000				Failure	Prediction error

Table E.7: Detailed data about the dynamic e-grocery task conducted with the AIC and including prediction errors - Part 2 of 2

# Test	Speed	Total time	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
1	1000	2,81 sec	0,85 sec	1,96 sec	Success	
2	1000	3,13 sec	0,78 sec	2,35 sec	Success	
3	1000				Failure	Cartesian reflex
4	1000				Failure	Prediction error
5	1000				Failure	Prediction error
6	1000	2,94 sec	0,92 sec	2,02 sec	Success	
7	1000				Failure	Cartesian reflex
8	1000				Failure	Prediction error
9	1000	3,38 sec	0,78 sec	2,6 sec	Success	
10	1000	2,91 sec	0,85 sec	2,06 sec	Success	
11	1000	3,27 sec	0,89 sec	2,38 sec	Success	
12	1000				Failure	Prediction error
13	1000				Failure	Prediction error
14	1000	2,81 sec	0,82 sec	1,99 sec	Success	
15	1000	3,23 sec	0,86 sec	2,36 sec	Success	
16	1000				Failure	Prediction error
17	1000	3,5 sec	0,86 sec	2,64 sec	Success	
18	1000	3,56 sec	0,83 sec	2,73 sec	Success	
19	1000	2,69 sec	0,8 sec	1,89 sec	Success	
20	1000	3,16 sec	0,77 sec	2,4 sec	Success	
21	1000	2,52 sec	0,8 sec	1,73 sec	Success	
22	1000				Failure	Prediction error
23	1000	2,8 sec	0,82 sec	1,99 sec	Success	
24	1000				Failure	Cartesian reflex
25	1000	3,31 sec	0,77 sec	2,54 sec	Success	
26	1000	2,88 sec	0,8 sec	2,07 sec	Success	
27	1000	2,72 sec	0,77 sec	1,96 sec	Success	
28	1000	3,38 sec	0,8 sec	2,58 sec	Success	
29	1000	2,48 sec	0,79 sec	1,69 sec	Success	
30	1000	3,33 sec	0,8 sec	2,53 sec	Success	
31	1500				Failure	Cartesian reflex
32	1500	10,62 sec	4,14 sec	6,48 sec	Success	
33	1500				Failure	Prediction error
34	1500				Failure	Prediction error
35	1500				Failure	Prediction error
36	1500	9,45 sec	3,09 sec	6,36 sec	Success	
37	1500	3,34 sec	0,78 sec	2,56 sec	Success	
38	1500				Failure	Cartesian reflex
39	1500	3,18 sec	0,75 sec	2,43 sec	Success	
40	1500	5,72 sec	2,88 sec	2,84 sec	Success	
41	1500				Failure	Prediction error
42	1500	2,77 sec	0,83 sec	1,94 sec	Success	
43	1500	10,11 sec	3,67 sec	6,45 sec	Success	
44	1500	5,38 sec	3,06 sec	2,33 sec	Success	
45	1500	3,35 sec	0,78 sec	2,56 sec	Success	

Table E.8: Detailed data about the dynamic e-grocery task conducted with the uAIC and including prediction errors - Part 1 of 2

# Test	Speed	Total time	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
46	1500				Failure	Prediction error
47	1500				Failure	Cartesian reflex
48	1500	2,72 sec	0,75 sec	1,98 sec	Success	
49	1500				Failure	Prediction error
50	1500	5,21 sec	4,19 sec	1,03 sec	Success	
51	1500	3,14 sec	0,82 sec	2,32 sec	Success	
52	1500	3,16 sec	0,77 sec	2,38 sec	Success	
53	1500	11,53 sec	4,99 sec	6,54 sec	Success	
54	1500	3,24 sec	0,76 sec	2,48 sec	Success	
55	1500	6,31 sec	3,52 sec	2,8 sec	Success	
56	1500	3,2 sec	0,76 sec	2,44 sec	Success	
57	1500	9,45 sec	3,77 sec	5,68 sec	Success	
58	1500	3,3 sec	0,78 sec	2,52 sec	Success	
59	1500				Failure	Prediction error
60	1500				Failure	Prediction error
61	2000				Failure	Failed to converge to set-point
62	2000	9,52 sec	3,35 sec	6,17 sec	Success	
63	2000				Failure	Prediction error
64	2000	6,07 sec	2,54 sec	3,53 sec	Success	
65	2000	8,28 sec	2,28 sec	6,0 sec	Success	
66	2000				Failure	Failed to converge to set-point
67	2000	9,92 sec	3,56 sec	6,36 sec	Success	
68	2000	4,08 sec	1,15 sec	2,93 sec	Success	
69	2000	9,98 sec	3,56 sec	6,42 sec	Success	
70	2000	5,98 sec	3,76 sec	2,22 sec	Success	
71	2000	10,51 sec	3,97 sec	6,53 sec	Success	
72	2000	7,53 sec	4,58 sec	2,95 sec	Success	
73	2000				Failure	Prediction error
74	2000	6,32 sec	3,43 sec	2,88 sec	Success	
75	2000	9,8 sec	3,48 sec	6,32 sec	Success	
76	2000	3,43 sec	0,87 sec	2,56 sec	Success	
77	2000				Failure	Prediction error
78	2000	4,88 sec	2,24 sec	2,64 sec	Success	
79	2000				Failure	Prediction error
80	2000	3,91 sec	1,64 sec	2,27 sec	Success	
81	2000	7,14 sec	3,96 sec	3,18 sec	Success	
82	2000	3,31 sec	1,06 sec	2,25 sec	Success	
83	2000				Failure	Prediction error
84	2000	9,51 sec	2,94 sec	6,58 sec	Success	
85	2000	10,16 sec	3,65 sec	6,52 sec	Success	
86	2000				Failure	Prediction error
87	2000	2,82 sec	0,79 sec	2,03 sec	Success	
88	2000	5,84 sec	3,56 sec	2,28 sec	Success	
89	2000				Failure	Prediction error
90	2000	10,01 sec	3,4 sec	6,62 sec	Success	

Table E.9: Detailed data about the dynamic e-grocery task conducted with the uAIC and including prediction errors - Part 2 of 2

# Test	Speed	Total time	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
1	1000	2,78 sec	1,11 sec	1,67 sec	Success	
2	1000	3,39 sec	1,48 sec	1,91 sec	Success	
3	1000				Failure	Failed to converge to set-point
4	1000	3,07 sec	1,14 sec	1,92 sec	Success	
5	1000				Failure	Failed to converge to set-point
6	1000				Failure	Failed to converge to set-point
7	1000				Failure	Failed to converge to set-point
8	1000	4,66 sec	1,13 sec	3,53 sec	Success	
9	1000				Failure	Failed to converge to set-point
10	1000	3,45 sec	1,52 sec	1,93 sec	Success	
11	1000	4,28 sec	1,55 sec	2,73 sec	Success	
12	1000	3,37 sec	1,4 sec	1,97 sec	Success	
13	1000	3,26 sec	1,4 sec	1,86 sec	Success	
14	1000	3,54 sec	1,63 sec	1,92 sec	Success	
15	1000				Failure	Failed to converge to set-point
16	1000	3,36 sec	1,4 sec	1,96 sec	Success	
17	1000	3,04 sec	1,1 sec	1,94 sec	Success	
18	1000				Failure	Failed to converge to set-point
19	1000	3,47 sec	1,46 sec	2,0 sec	Success	
20	1000	3,92 sec	1,58 sec	2,35 sec	Success	
21	1000	2,99 sec	1,0 sec	1,98 sec	Success	
22	1000				Failure	Failed to converge to set-point
23	1000	3,0 sec	1,13 sec	1,87 sec	Success	
24	1000	2,86 sec	1,07 sec	1,79 sec	Success	
25	1000	3,1 sec	1,19 sec	1,91 sec	Success	
26	1000				Failure	Cartesian reflex
27	1000				Failure	Failed to converge to set-point
28	1000				Failure	Failed to converge to set-point
29	1000				Failure	Failed to converge to set-point
30	1000	3,54 sec	1,48 sec	2,06 sec	Success	
31	1500	3,99 sec	1,7 sec	2,29 sec	Success	
32	1500	3,3 sec	1,38 sec	1,91 sec	Success	
33	1500	3,13 sec	1,28 sec	1,86 sec	Success	
34	1500				Failure	Failed to converge to set-point
35	1500	3,0 sec	1,13 sec	1,87 sec	Success	
36	1500				Failure	Failed to converge to set-point
37	1500				Failure	Failed to converge to set-point
38	1500				Failure	Failed to converge to set-point
39	1500	3,28 sec	1,29 sec	1,99 sec	Success	
40	1500				Failure	Failed to converge to set-point
41	1500	3,17 sec	1,24 sec	1,93 sec	Success	
42	1500				Failure	Failed to converge to set-point
43	1500	3,16 sec	1,26 sec	1,9 sec	Success	
44	1500				Failure	Failed to converge to set-point
45	1500				Failure	Failed to converge to set-point

Table E.10: Detailed data about the dynamic e-grocery task conducted with the MRAC and excluding prediction errors - Part 1 of 2

# Test	Speed	Total time	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
46	1500	3,13 sec	1,24 sec	1,89 sec	Success	
47	1500	2,88 sec	1,11 sec	1,77 sec	Success	
48	1500	2,86 sec	1,11 sec	1,75 sec	Success	
49	1500	3,02 sec	1,1 sec	1,91 sec	Success	
50	1500	2,98 sec	1,06 sec	1,92 sec	Success	
51	1500	3,27 sec	1,41 sec	1,86 sec	Success	
52	1500	3,14 sec	1,19 sec	1,95 sec	Success	
53	1500				Failure	Failed to converge to set-point
54	1500				Failure	Failed to converge to set-point
55	1500	2,87 sec	1,03 sec	1,85 sec	Success	
56	1500				Failure	Failed to converge to set-point
57	1500	2,96 sec	1,06 sec	1,9 sec	Success	
58	1500				Failure	Failed to converge to set-point
59	1500	2,86 sec	1,05 sec	1,82 sec	Success	
60	1500				Failure	Failed to converge to set-point
61	2000				Failure	Failed to converge to set-point
62	2000				Failure	Failed to converge to set-point
63	2000				Failure	Failed to converge to set-point
64	2000	3,91 sec	1,45 sec	2,47 sec	Success	
65	2000				Failure	Failed to converge to set-point
66	2000	3,45 sec	1,2 sec	2,24 sec	Success	
67	2000	2,94 sec	1,1 sec	1,84 sec	Success	
68	2000	4,35 sec	1,9 sec	2,45 sec	Success	
69	2000				Failure	Failed to converge to set-point
70	2000				Failure	Failed to converge to set-point
71	2000				Failure	Failed to converge to set-point
72	2000				Failure	Failed to converge to set-point
73	2000				Failure	Failed to converge to set-point
74	2000	3,28 sec	1,14 sec	2,14 sec	Success	
75	2000				Failure	Failed to converge to set-point
76	2000				Failure	Failed to converge to set-point
77	2000	3,1 sec	1,13 sec	1,97 sec	Success	
78	2000				Failure	Failed to converge to set-point
79	2000	3,42 sec	1,18 sec	2,24 sec	Success	
80	2000	2,86 sec	1,05 sec	1,82 sec	Success	
81	2000	2,91 sec	1,2 sec	1,7 sec	Success	
82	2000	3,2 sec	1,24 sec	1,95 sec	Success	
83	2000	2,7 sec	0,94 sec	1,76 sec	Success	
84	2000				Failure	Failed to converge to set-point
85	2000				Failure	Failed to converge to set-point
86	2000				Failure	Failed to converge to set-point
87	2000				Failure	Failed to converge to set-point
88	2000	2,99 sec	1,07 sec	1,92 sec	Success	
89	2000	3,25 sec	1,1 sec	2,15 sec	Success	
90	2000				Failure	Failed to converge to set-point

Table E.11: Detailed data about the dynamic e-grocery task conducted with the MRAC and excluding prediction errors - Part 2 of 2

# Test	Speed	Total time	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
1	1000	5,865	1,91	3,955	Success	
2	1000	5,836	1,984	3,852	Success	
3	1000	0			Failure	Failed to converge to set-point
4	1000	5,799	1,94	3,859	Success	
5	1000	0			Failure	Failed to converge to set-point
6	1000	5,82	1,948	3,872	Success	
7	1000	5,803	1,931	3,872	Success	
8	1000	5,472	1,936	3,536	Success	
9	1000	5,803	1,94	3,863	Success	
10	1000	5,726	1,951	3,775	Success	
11	1000	0			Failure	Failed to converge to set-point
12	1000	0			Failure	Failed to converge to set-point
13	1000	0			Failure	Failed to converge to set-point
14	1000	5,963	2,064	3,899	Success	
15	1000	5,78	1,951	3,829	Success	
16	1000	5,903	1,925	3,978	Success	
17	1000	5,817	1,978	3,839	Success	
18	1000	5,749	1,944	3,805	Success	
19	1000	5,779	1,907	3,872	Success	
20	1000	5,734	1,963	3,771	Success	
21	1000	5,839	1,903	3,936	Success	
22	1000	5,799	1,928	3,871	Success	
23	1000	5,827	1,968	3,859	Success	
24	1000				Failure	Failed to converge to set-point
25	1000	5,719	1,912	3,807	Success	
26	1000	5,916	1,94	3,976	Success	
27	1000	5,913	1,96	3,953	Success	
28	1000				Failure	Cartesian reflex
29	1000	5,778	1,895	3,883	Success	
30	1000	5,78	1,868	3,912	Success	
31	1500	5,89 sec	1,95 sec	3,94 sec	Success	
32	1500	5,6 sec	1,91 sec	3,68 sec	Success	
33	1500	5,81 sec	1,91 sec	3,91 sec	Success	
34	1500	5,94 sec	1,94 sec	4,0 sec	Success	
35	1500	5,88 sec	1,9 sec	3,98 sec	Success	
36	1500	6,46 sec	1,94 sec	4,52 sec	Success	
37	1500	5,95 sec	2,06 sec	3,89 sec	Success	
38	1500				Failure	Failed to converge to set-point
39	1500				Failure	Failed to converge to set-point
40	1500	6,0 sec	1,96 sec	4,04 sec	Success	
41	1500	5,8 sec	1,87 sec	3,92 sec	Success	
42	1500	5,99 sec	2,01 sec	3,98 sec	Success	
43	1500	5,8 sec	1,94 sec	3,86 sec	Success	
44	1500	5,71 sec	1,86 sec	3,85 sec	Success	
45	1500	6,01 sec	1,95 sec	4,07 sec	Success	

Table E.12: Detailed data about the dynamic e-grocery task conducted with the AIC and excluding prediction errors - Part 1 of 2

# Test	Speed	Total time	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
46	1500				Failure	Failed to converge to set-point
47	1500	5,96 sec	1,96 sec	4,0 sec	Success	
48	1500	5,9 sec	1,91 sec	4,0 sec	Success	
49	1500	5,94 sec	1,97 sec	3,98 sec	Success	
50	1500	5,61 sec	1,95 sec	3,66 sec	Success	
51	1500	5,87 sec	1,96 sec	3,92 sec	Success	
52	1500	6,05 sec	1,96 sec	4,09 sec	Success	
53	1500	5,83 sec	1,88 sec	3,96 sec	Success	
54	1500	5,82 sec	1,92 sec	3,9 sec	Success	
55	1500	5,84 sec	1,9 sec	3,94 sec	Success	
56	1500				Failure	Failed to converge to set-point
57	1500	5,81 sec	1,9 sec	3,9 sec	Success	
58	1500	5,79 sec	1,91 sec	3,88 sec	Success	
59	1500				Failure	Cartesian reflex
60	1500	5,98 sec	2,01 sec	3,96 sec	Success	
61	2000	5,67 sec	1,8 sec	3,87 sec	Success	
62	2000				Failure	Failed to converge to set-point
63	2000	5,88 sec	1,96 sec	3,92 sec	Success	
64	2000				Failure	Failed to converge to set-point
65	2000				Failure	Failed to converge to set-point
66	2000	6,98 sec	2,9 sec	4,09 sec	Success	
67	2000	6,94 sec	3,07 sec	3,87 sec	Success	
68	2000				Failure	Failed to converge to set-point
69	2000				Success	
70	2000				Failure	Failed to converge to set-point
71	2000	8,03 sec	3,65 sec	4,38 sec	Success	
72	2000				Failure	Failed to converge to set-point
73	2000	5,88 sec	1,9 sec	3,98 sec	Success	
74	2000				Failure	Failed to converge to set-point
75	2000	5,93 sec	1,99 sec	3,94 sec	Success	
76	2000	6,45 sec	2,48 sec	3,97 sec	Success	
77	2000				Failure	Failed to converge to set-point
78	2000	5,9 sec	1,96 sec	3,94 sec	Success	
79	2000				Failure	Failed to converge to set-point
80	2000	5,82 sec	1,95 sec	3,88 sec	Success	
81	2000	8,25 sec	4,27 sec	3,98 sec	Success	
82	2000	8,85 sec	4,76 sec	4,08 sec	Success	
83	2000				Failure	Failed to converge to set-point
84	2000	7,15 sec	3,16 sec	3,99 sec	Success	
85	2000				Failure	Failed to converge to set-point
86	2000	7,43 sec	3,46 sec	3,97 sec	Success	
87	2000	8,34 sec	4,34 sec	4,0 sec	Success	
88	2000				Failure	Failed to converge to set-point
89	2000				Failure	Failed to converge to set-point
90	2000				Failure	Failed to converge to set-point

Table E.13: Detailed data about the dynamic e-grocery task conducted with the AIC and excluding prediction errors - Part 2 of 2

# Test	Speed	Total time	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
1	1000	3,37 sec	0,89 sec	2,48 sec	Success	
2	1000				Failure	Cartesian reflex
3	1000	3,53 sec	0,88 sec	2,64 sec	Success	
4	1000	3,46 sec	0,87 sec	2,59 sec	Success	
5	1000	3,26 sec	0,86 sec	2,4 sec	Success	
6	1000	3,2 sec	0,85 sec	2,35 sec	Success	
7	1000	3,38 sec	0,82 sec	2,56 sec	Success	
8	1000	3,34 sec	0,88 sec	2,47 sec	Success	
9	1000	2,71 sec	0,9 sec	1,82 sec	Success	
10	1000				Failure	Cartesian reflex
11	1000	3,1 sec	0,79 sec	2,3 sec	Success	
12	1000	3,07 sec	0,8 sec	2,27 sec	Success	
13	1000	3,12 sec	0,76 sec	2,35 sec	Success	
14	1000	3,34 sec	0,82 sec	2,52 sec	Success	
15	1000	2,96 sec	0,85 sec	2,11 sec	Success	
16	1000	3,43 sec	0,84 sec	2,58 sec	Success	
17	1000	3,29 sec	0,82 sec	2,47 sec	Success	
18	1000	3,01 sec	0,8 sec	2,21 sec	Success	
19	1000	3,31 sec	0,85 sec	2,47 sec	Success	
20	1000	3,17 sec	0,79 sec	2,38 sec	Success	
21	1000	3,15 sec	0,81 sec	2,34 sec	Success	
22	1000	2,99 sec	0,74 sec	2,25 sec	Success	
23	1000	3,43 sec	0,85 sec	2,58 sec	Success	
24	1000	3,02 sec	0,87 sec	2,15 sec	Success	
25	1000	3,29 sec	0,81 sec	2,48 sec	Success	
26	1000	3,32 sec	0,78 sec	2,54 sec	Success	
27	1000	3,2 sec	0,81 sec	2,39 sec	Success	
28	1000	3,1 sec	0,8 sec	2,31 sec	Success	
29	1000	3,13 sec	0,84 sec	2,29 sec	Success	
30	1000	3,1 sec	0,86 sec	2,24 sec	Success	
31	1500	3,0 sec	0,91 sec	2,1 sec	Success	
32	1500	3,43 sec	0,84 sec	2,59 sec	Success	
33	1500	2,9 sec	0,72 sec	2,18 sec	Success	
34	1500	2,92 sec	0,96 sec	1,96 sec	Success	
35	1500	3,32 sec	0,85 sec	2,47 sec	Success	
36	1500	0,0 sec			Failure	Failed to converge to set-point
37	1500	0,0 sec			Failure	Failed to converge to set-point
38	1500	3,37 sec	0,86 sec	2,51 sec	Success	
39	1500	3,2 sec	0,81 sec	2,39 sec	Success	
40	1500	2,99 sec	0,8 sec	2,18 sec	Success	
41	1500				Failure	Failed to converge to set-point
42	1500	3,09 sec	0,75 sec	2,34 sec	Success	
43	1500	2,72 sec	0,82 sec	1,9 sec	Success	
44	1500	3,1 sec	0,79 sec	2,32 sec	Success	
45	1500	2,89 sec	0,83 sec	2,06 sec	Success	

Table E.14: Detailed data about the dynamic e-grocery task conducted with the uAIC and excluding prediction errors - Part 1 of 2

# Test	Speed	Total time	Time to grasp the object	Time needed to go to placing position	Result	Failure reason
46	1500	3,1 sec	0,78 sec	2,32 sec	Success	
47	1500	3,45 sec	0,86 sec	2,59 sec	Success	
48	1500	3,39 sec	0,84 sec	2,54 sec	Success	
49	1500	3,1 sec	0,78 sec	2,32 sec	Success	
50	1500	3,35 sec	0,82 sec	2,53 sec	Success	
51	1500	3,26 sec	0,78 sec	2,48 sec	Success	
52	1500	3,27 sec	0,84 sec	2,44 sec	Success	
53	1500	3,39 sec	0,85 sec	2,54 sec	Success	
54	1500	3,29 sec	0,78 sec	2,51 sec	Success	
55	1500	3,37 sec	0,82 sec	2,55 sec	Success	
56	1500				Failure	Failed to converge to set-point
57	1500	3,33 sec	0,82 sec	2,51 sec	Success	
58	1500	3,29 sec	0,82 sec	2,47 sec	Success	
59	1500	3,48 sec	0,8 sec	2,68 sec	Success	
60	1500				Failure	Cartesian reflex
61	2000	3,77 sec	0,91 sec	2,86 sec	Success	
62	2000	3,23 sec	0,85 sec	2,38 sec	Success	
63	2000	3,42 sec	0,84 sec	2,59 sec	Success	
64	2000	3,47 sec	0,89 sec	2,57 sec	Success	
65	2000	3,19 sec	0,85 sec	2,34 sec	Success	
66	2000	3,22 sec	0,83 sec	2,39 sec	Success	
67	2000	3,26 sec	0,82 sec	2,44 sec	Success	
68	2000	3,18 sec	0,8 sec	2,38 sec	Success	
69	2000	3,23 sec	0,83 sec	2,4 sec	Success	
70	2000	3,42 sec	0,82 sec	2,6 sec	Success	
71	2000	3,42 sec	0,83 sec	2,58 sec	Success	
72	2000	3,13 sec	0,77 sec	2,36 sec	Success	
73	2000	3,44 sec	0,82 sec	2,62 sec	Success	
74	2000	3,28 sec	0,84 sec	2,44 sec	Success	
75	2000	3,28 sec	0,82 sec	2,46 sec	Success	
76	2000	3,42 sec	0,85 sec	2,58 sec	Success	
77	2000	3,09 sec	0,78 sec	2,32 sec	Success	
78	2000	3,14 sec	0,78 sec	2,36 sec	Success	
79	2000	3,48 sec	0,78 sec	2,71 sec	Success	
80	2000	3,5 sec	0,88 sec	2,62 sec	Success	
81	2000	2,67 sec	0,87 sec	1,8 sec	Success	
82	2000	3,24 sec	0,8 sec	2,44 sec	Success	
83	2000	3,08 sec	0,78 sec	2,31 sec	Success	
84	2000	3,24 sec	0,82 sec	2,42 sec	Success	
85	2000	3,38 sec	0,81 sec	2,56 sec	Success	
86	2000	3,34 sec	0,82 sec	2,52 sec	Success	
87	2000	3,39 sec	0,85 sec	2,54 sec	Success	
88	2000	3,02 sec	0,8 sec	2,22 sec	Success	
89	2000				Failure	Failed to converge to set-point
90	2000	3,05 sec	0,87 sec	2,18 sec	Success	

Table E.15: Detailed data about the dynamic e-grocery task conducted with the uAIC and excluding prediction errors - Part 2 of 2

F Startup procedure for the Franka Emika Panda Robot

1. Turn on the black controller box below the table;
2. Disengage the red-yellow Emergency Stop Button labeled as the E-Stop button (Figure F.1);
3. The lights at the base of the robot should now be blinking yellow;
4. Open Chrome and navigate to https://172.16.0.3/desk/;
5. On the website, navigate to 'Joints' on the right-hand side and press the unlock button. Then, in the pop-up window press 'open'. You will start hearing the joints opening. The light should now be white or blue;
6. If the light is blue press down the black "Mode" button (Figure F.2);
7. You can now press the two buttons on the side of the end effector (the two that are right across from each other) to move the robot arm freely. Bring the robot to its default pose (Figure F.2). This is important in order to correctly start the various controllers;
8. Disengage the Mode Button by rotating clockwise. The light should now be blue;
9. On the website, navigate to the menu and press "Activate FCI". You are now able to send commands and receive information to and from the robot over ROS.

Figure F.1: Emergency button

Figure F.2: Mode button

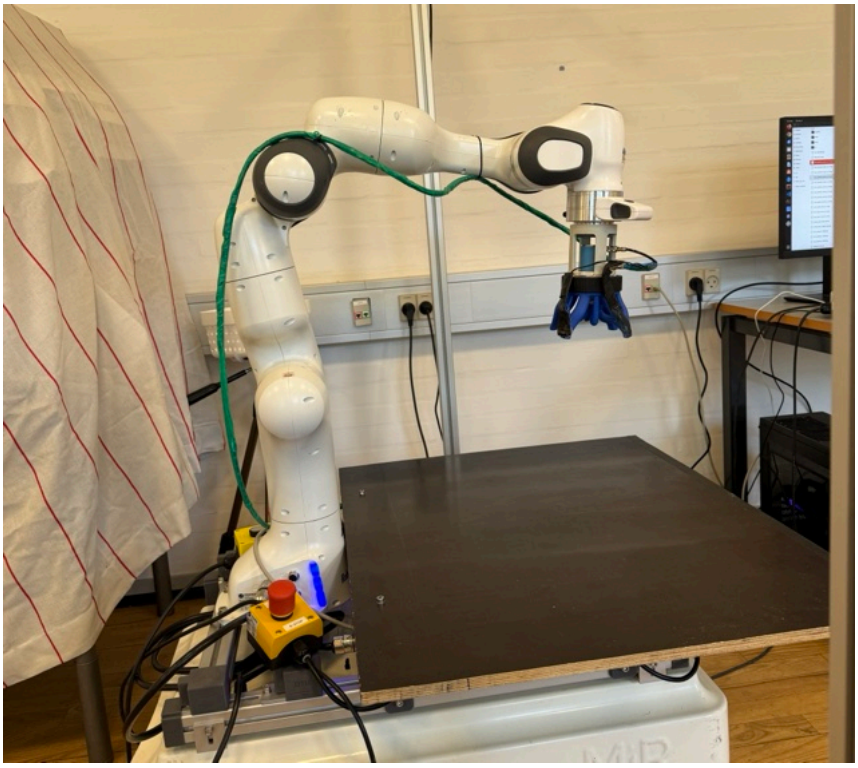


Figure F.3: Default pose used during the startup of the controllers

F.1 Launching the e-grocery mission

1. First open the script ”object_tracking_static.py” in the ”object_detection” package and run it;
2. Place an object on the (e.g. tennis ball). Two screen should pop-up. One with the depth-image and one with the rgb-image. On the latter you should see some blue and white lines, if you don’t see them you should un-comment line 264. If you are running the dynamic e-grocery task with the conveyor belt you should un-comment lines 253-254;
3. Align the white horizontal line with the upper (from camera view, facing the robot

base) side of the bolts (Figure F.4). If needed align the conveyor belt whit the two white horizontal lines on the bottom;

4. Once the camera frame is aligned to the robot frame you can stop the script with CTRL-C;
5. If needed run the conveyor belt at the desired speed;
6. On the computer navigate to the ws:
    - cd Gabriele_thesis/panda_ws
7. Open 4 more terminals and in all of them do:
    - source ./devel/setup.bash
8. In each terminal run:
    - roscore
    - roslaunch franka_aic AIC_controller.launch
9. If you are running the static e-grocery then place the ball in a random position and run, in the other terminal:
    - roslaunch move_panda egrocery_static_task.launch (for the static e-grocery task) or...
10. If you are running the dynamic e-grocery then run in the other terminal:
    -
    - roslaunch move_panda egrocery_dynamic_task.launch (for the dynamic e-grocery task with the conveyor belt)
11. then place the ball at the very start of the conveyor belt, on the left of the robot, by paying attention in placing the ball by grasping it from the sides and not placing your hand on top of it (this is to guarantee the camera will pick the right distance reading).

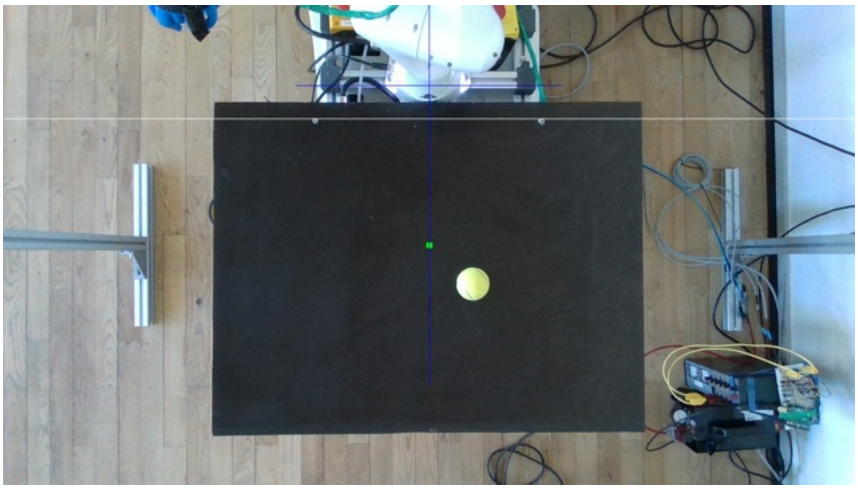


Figure F.4: Calibration lines to properly align the camera to the robot

Technical
University of
Denmark

Elektrovej, Building 326
2800 Kgs. Lyngby
Tlf. 4525 1700

https://electro.dtu.dk/