# Control of a 7-DOF MRI Compatible Robot Arm

*Naichen Zhao*

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2025-100
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-100.html

May 16, 2025

Acknowledgement

I am extremely grateful to Professor Ronald Fearing for his mentorship while I was working as a part of the Biomimetic Millisystems Lab. None of this work would have been possible without his guidance. I also appreciate the work of our collaborators on the overarching MRI Robot project, especially professor Michael Lustig who has provided advice on MRI compatibility and also has agreed to be the second reader of this report.

I would like to thank Dr Binghan He for his work leading the robot arm development and mechanical design. I would also like to thank Alfredo De Goyeneche for his assistance with all of the MRI testing. I am also grateful for all my peers in the Biomimetic Millisystems Lab, notably Charles Paxson, Martin Zeng, and David Guo who also worked on the robot arm.

**Acknowledgments**

I am extremely grateful to Professor Ronald Fearing for his mentorship while I was working as a part of the Biomimetic Millisystems Lab. None of this work would have been possible without his guidance. I also appreciate the work of our collaborators on the overarching MRI Robot project, especially professor Michael Lustig who has provided advice on MRI compatibility and also has agreed to be the second reader of this report.

I would like to thank Dr Binghan He for his work leading the robot arm development and mechanical design. I would also like to thank Alfredo De Goyeneche for his assistance with all of the MRI testing. I am also grateful for all my peers in the Biomimetic Millisystems Lab, notably Charles Paxson, Martin Zeng, and David Guo who also worked on various parts of the robot arm.

# Control of a 7-DOF MRI Compatible Robot Arm

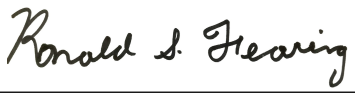
by Naichen Zhao

## Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

**Committee**

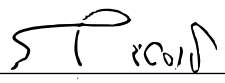


Professor Ronald S. Fearing
Research Advisor

May 15, 2025

(Date)

⋆ ⋆ ⋆ ⋆ ⋆ ⋆ ⋆



Professor Michael Lustig
Second Reader

May 16, 2025

(Date)

Abstract

Control of a 7-DOF MRI Compatible Robot Arm

by

Naichen Zhao

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Ronald S. Fearing, Chair

Professor Michael Lustig, Co-chair

This work describes the structure and testing of the software, firmware, and electronics used to control a custom built MRI compatible 7DOF robot arm. The arm itself is designed to maneuver a Transcranial Magnetic Stimulation (TMS) coil around a patient's head within a MRI bore, allowing for precise positioning of the TMS coil.

The Arm is constructed using MRI compatible materials, driven by 7 Ultrasonic motors, each with its own Series Elastic Actuator (SEA), allowing for torque sensing on each joint. A microcontroller oversees the arm's low-level functions, running firmware written using the Lingua Franca (LF) framework, while the higher level planning is performed on a Linux computer running ROS2. The system is capable of tacking the force experienced on each joint while performing gravity compensation to filter out any gravitational load. The controller is also shown using position-based PID control to maneuver the arm to various positions while operating with a variety of end-effector weights.

This project's documents (including PCB schematics/layouts, microcontroller firmware, and ROS packages) can be found at: [https://github.com/biomimetics/MRIRobotProject].

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 MRI Robot Project

Transcranial Magnetic Stimulation (TMS) is a medical technique which involves using magnetic fields to stimulate neurons in a patient's brain, allowing for the treatment of various disorders. However, the usage of such methods is still an active area of research, especially in terms of properly measuring the brain's responses to TMS pulses. This project is part of an over-arching effort to develop a robot arm which would help perform TMS operations within a MRI machine, allowing for greater insight into its impacts on the human brain [1].

The mechanical design for the arm was previously completed by Dr. Bingham He while working under Professor Ronald Fearing in Berkeley's Biomimetic Millisystems Lab. This report will build on his work, developing the robot's planning software, firmware, and electronics. It will also detail the initial integration and testing of the robot arm.

## 1.2 Related Work

### Robot-assisted TMS operations

Robot-assisted positioning has been recognized as a useful tool in the area of TMS research [2] [3] [4] generally leading to better positioning accuracy. However, these tests are performed outside of an MRI machine, with the patient first getting an MRI scan, and then the researchers later using that data to perform TMS operations. This means we cannot use the MRI to directly measure the brain's immediate response to the TMS pulses, making it more difficult to understand its full impact. The best option would be to place the robot within the MRI bore, taking advantage of better positioning while gaining greater insight into brain activity.

### MRI Robots

Designing robots which can function within a MRI bore is quite challenging, as there are strict limitations on what sensing, actuation, and materials are available [5]. Notably, standard magnetic driven DC motors cannot be used, with most robots powered by either pneumatics/hydraulics, piezoelectric motors, or more non-conventional methods of movement [6] [7] [8] [9]. These robots are usually designed to perform surgical tasks within a MRI bore [10] [11] [12] with design optimizations focusing on being lightweight and precise. However, positioning a TMS coil requires a significantly more heavy duty arm. The TMS coil is expected to weight over 1kg, meaning the robot must be robust enough to accurately manipulate such a weight within the bore and around a patient's head.

Sensing withing a MRI machine is also extremely important, especially when operating in close proximity to a patient. Force-based control is a common method of ensuring patient safety, with there being several papers that describe MRI-compatible force sensors [13]. Specifically, series elastic actuators have been used to enable force-feedback on non-backdrivable motors [14], allowing for better compliance to be built into the system, increasing the level of safety.

## 1.3 System Overview

The primary challenge of this project is to ensure the robot is fully MRI compatible, both in terms of being able to operate within a MRI bore and minimizing its impact on scanner noise. Thus, the robot arm itself is primarily built using plastic, with small amounts of brass and aluminum. Although non-ferromagnetic metals are allowed inside a MRI machine, they can cause artifacts in the scanning process, meaning their usage should ideally be minimized.

For joint actuation, we used ultrasonic motors developed by Tekceleo. These are one of the few commercially available MRI compatible motors, as conventional DC, BLDC, and stepper motors are built using magnets. However, ultrasonic motors are non-backdrivable, meaning compliance must be introduced elsewhere to ensure safety when operating in close proximity to a patient. Each joint is fitted with a Series Elastic Actuator (SEA) [15] which allow us to measure the torque experienced at each joint. These sensors consist of two encoders on either side of an elastic element (Fig. 1.1), using the relative displacement of the encoders to solve for the torque. this allows us to track the measured torque at each angle, letting us know if the robot has collided with a wall or object.

All electronics are yo be placed outside the MRI room in an adjacent control room (Fig. 1.2). The PCBs and electronics contain a number of magnetic materials - like inductors used for the ultrasonic motor drivers - meaning they must be kept far away from the MRI. All the motor control and sensor signals must be routed through 15+ meter cables from the control room all the way to the robot arm. All wires must be properly shielded and any sensor readings filtered to minimize any induced noise generated by MRI scans.

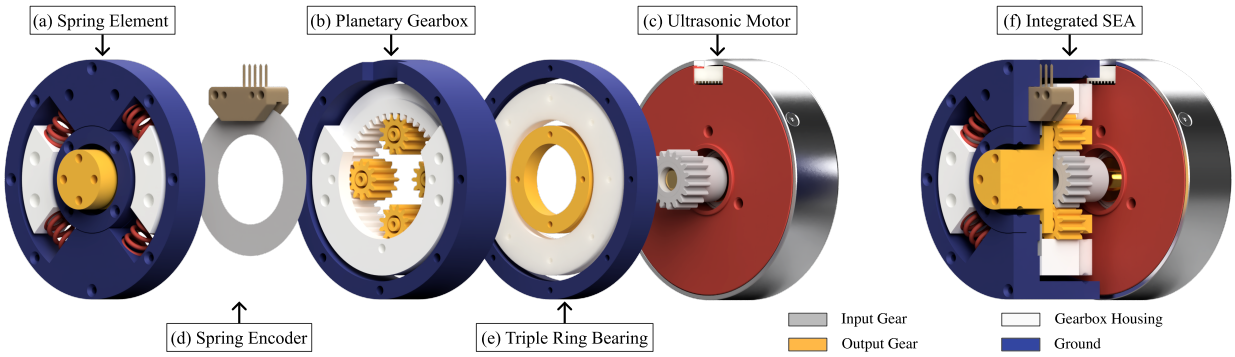



Figure 1.1: Diagram of the Series Elastic Actuator taken from [15]

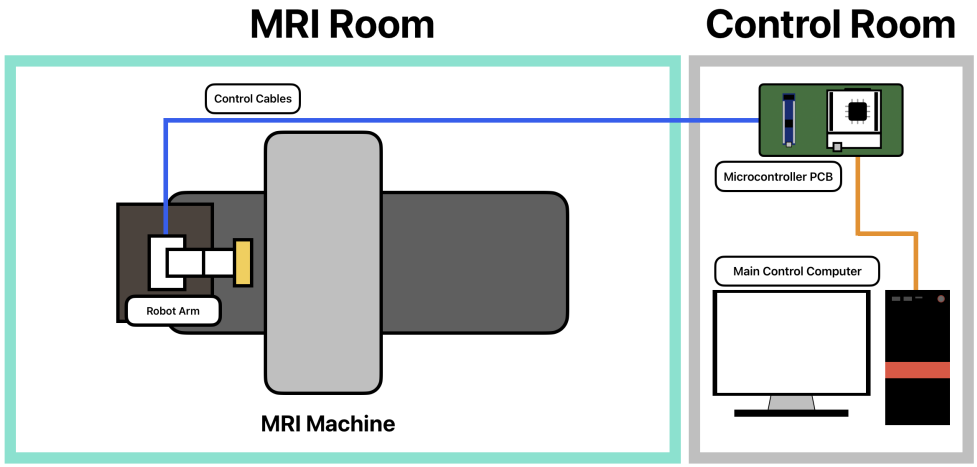



Figure 1.2: MRI robot room setup

The robot's control stack consists of two components: A Linux-based computer used for high-level planning, and a STM32 F446RE microcontroller for low-level operations. The main computer handles planning and robot dynamics, getting the target position from the user and generating the desired path and necessary joint angles. Meanwhile, the low-level controller takes those position commands and moves the robot to the desired location, monitoring the arm's state to ensure safety. Other sensing interfaces are also used, including MRI markers which allow us to get absolute positioning of the end-effector within the MRI bore [16].

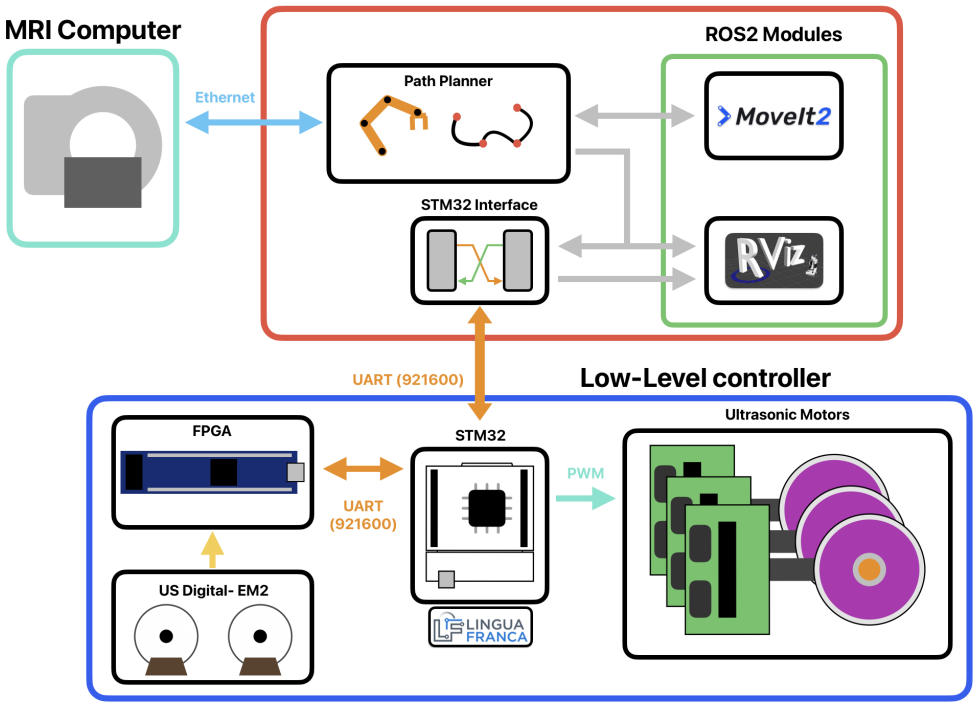



Figure 1.3: Robot system block diagram

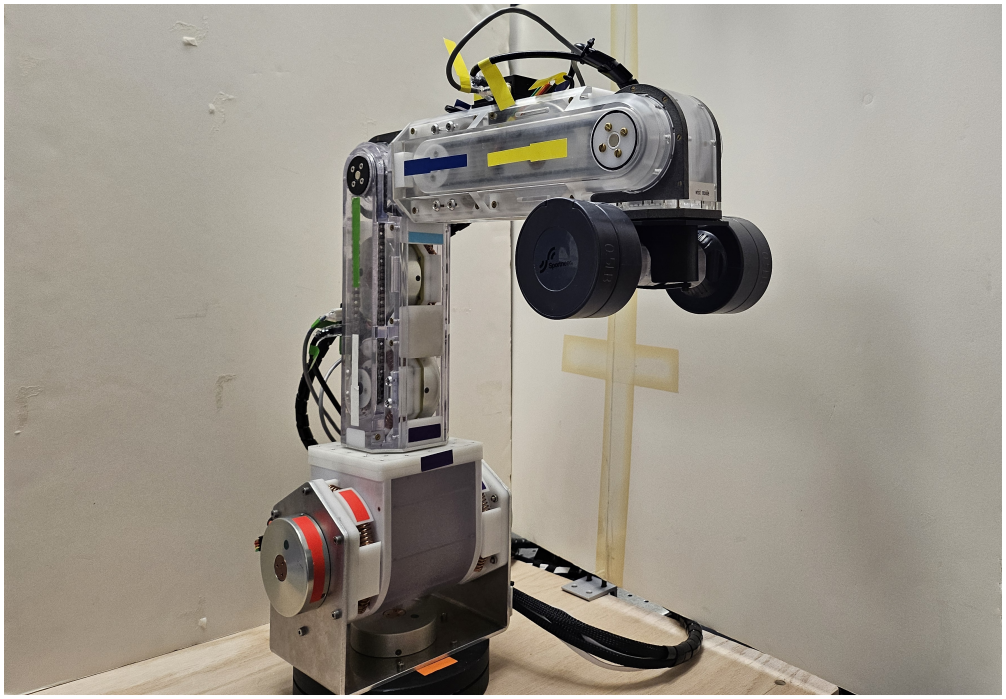


Figure 1.4: Robot arm in lab

# Chapter 2

# Software

## 2.1 Mechanical Integration

Since this project involves using a custom robot arm, the first step was to integrate the robot's physical properties into our dynamics software. This involves converting the robot arm's CAD representation into a file format format which is understood by the software stack.

### Universal Robot Description Format

A Universal Robot Description Format (URDF) file is an .xml file specifying a robot's properties, including: joints, sensors, movement range, and mass. It is used as a baseline for inverse kinematics solving, collision detection, and gravity compensation. The URDF file also points to meshes of the robot's joints which can be used for state visualization.

### URDF Exporting

To generate a URDF, we use the **Solidworks URDF exporter** toolchain. This involves selecting the robot's joints and defining their relative positions and movement constraints. The robot has 7 total joints as seen in Table 2.1. The mass properties and moments of inertia are also automatically generated through this process, though the joint masses were also verified against physical measurements to ensure accuracy.

| Joint | type |
|---|---|
| Shoulder_Base | Revolute |
| Shoulder_Joint | Revolute |
| Arm_Upper | Revolute |
| Elbow | Revolute |
| Arm_Lower | Revolute |
| Wrist | Revolute |
| TMS_Coil | Revolute |

Table 2.1: Robot arm joints

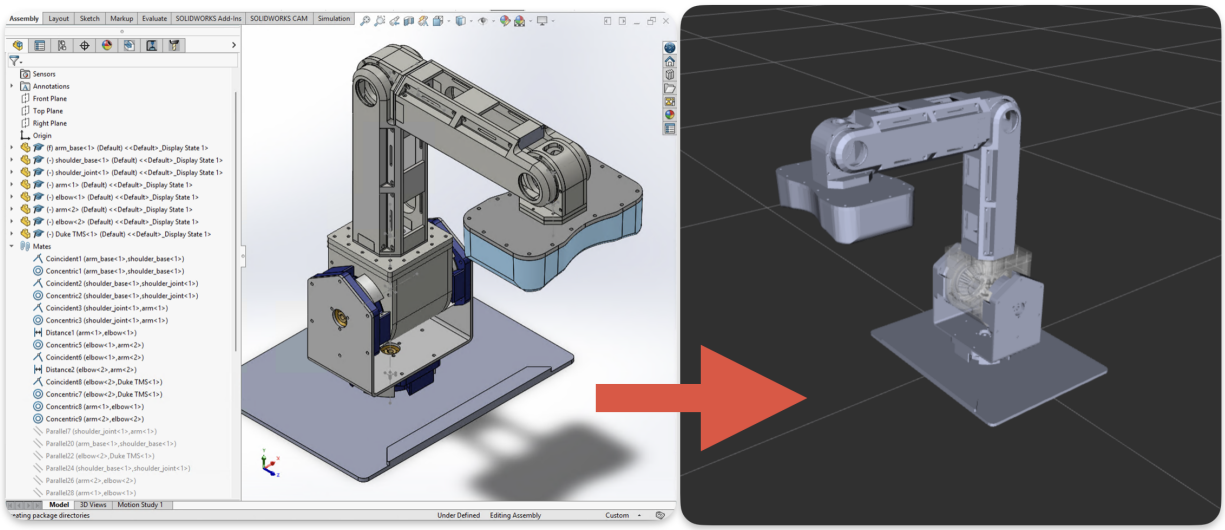


Figure 2.1: Solidworks model (left) and corresponding URDF (right)

## 2.2 ROS2

The software can be found here: [https://github.com/biomimetics/MRIRobot_ROS.git].

We decided to use ROS2 framework to construct the robot's high-level planner. ROS2 (Robot Operating System 2) [17] is an open source framework for deploying software onto robotics platforms. the ROS2 instance runs on our high-level control computer, using a using a USB-UART serial interface to communicate with a STM32 microcontroller. Specifically, the robot arm will be using *ROS2 Jazzy* paired with the latest version of Ubuntu Linux - *Ubuntu 24.04*.

The robot's software architecture can be broken up into two components, as seen in Fig. 2.2.

1. The **Arm Controller** which receives the target end-effector location and performs the necessary path planning.

2. The **STM32 Bridge** which communicates with the STM32, visualizes the current state, and performs the gravity compensation.

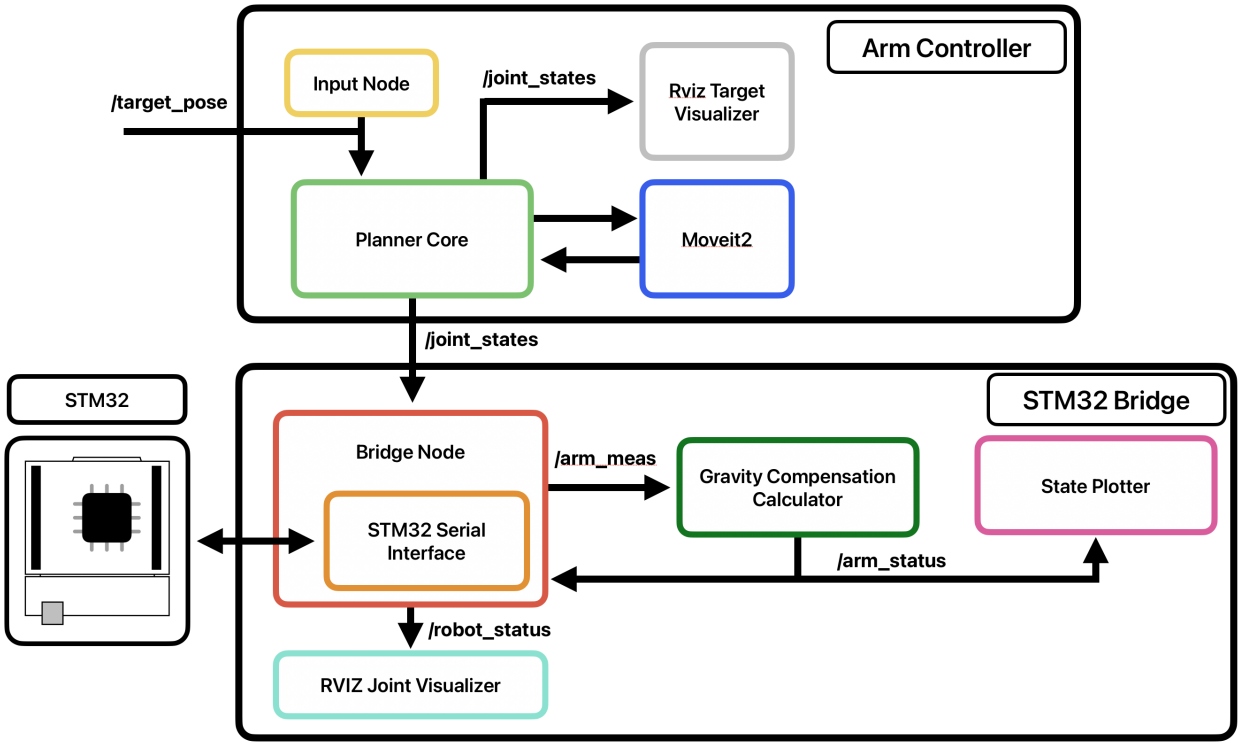



Figure 2.2: ROS2 software block diagram

## 2.3 Arm Controller

The **Arm Controller** block acts as the main core of the high-level planner. It takes in a desired end-effector pose and makes the relevant calls to Moveit2's inverse kinematics and path planner libraries to generate a desired trajectory for the arm.

### Moveit2

Moveit2 is an open source ROS2 compatible library which incorporates control, kinematics, navigation, and planning. It is compatible with multiple standard robot arm platforms and also allows the user to import their own robot description (Using a URDF file) through their import wizard.

Moveit2 can be called through the **MoveItPy** python library. The user provides the current and target poses in workspace coordinates, and MoveIt will generate a planned trajectory to move to the desired location. Internally, Moveit2 performs collision detection, ensuring the robot does not hit either itself or the surrounding environment. It will also create a

visualization of the intended path through rviz which the user can visually inspect for abnormalities.

### rviz

Rviz is ROS2's built in 3D visualization interface. It is has support for multiple data types, being able to render basic geometries, transform frames, point clouds, and custom STL models. rviz is also able to read the robot's URDF, directly importing the relevant meshes to display the robot's state, alongside any relevant environmental objects.

## Planner Core

The main high-level controller is found within the *planner Core* Node. To initiate a movement, a desired position is published to the **target_pose** ROS2 topic, either from the Input Node (for internal testing purposes) or from an external source (This is intended to be the MRI control computer). The *Planner Core* Node will then send a request to Moveit2 with the robot's current position, as well as the desired end-effector pose. The returned path is compared against predefined movement criteria, ensuring the path stays within the robot's joint limits. If it fails, a new request is generated, with this process repeating until a satisfactory trajectory is found. The plan is then executed, with joint positions being sent published to the **/joint_states** topic.

The planner core also adds relevant scene objects. Since the robot will be operating inside a MRI bore, it is important to define the MRI's walls so that the Moveit2 can plan around it. We also insert an estimate of where the patient's head will be so that the robot arm also avoid that area.

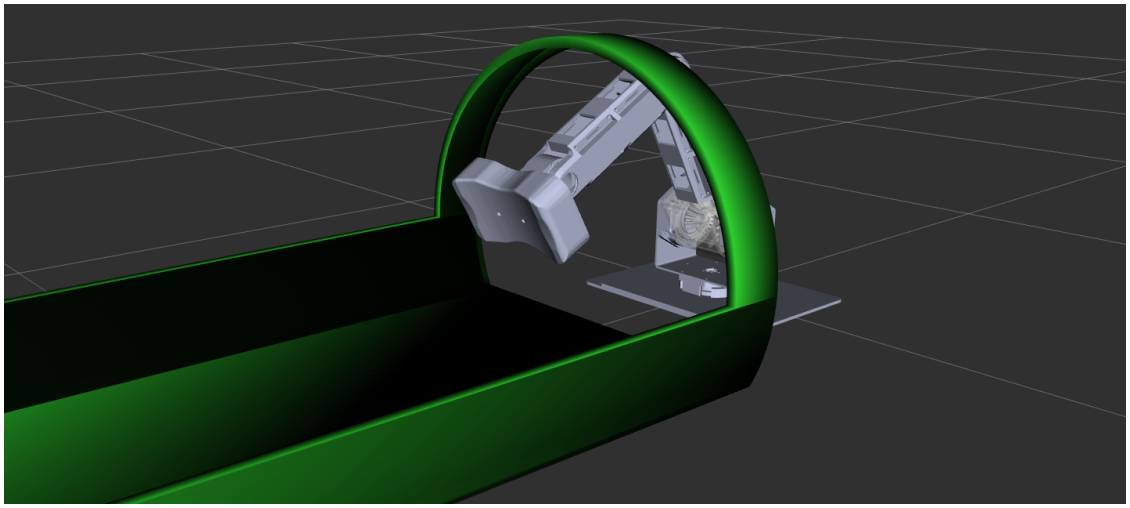


Figure 2.3: rviz scene with robot arm inside MRI bore model

## 2.4 STM32 Bridge

### Serial Interface

To communicate between the STM32 and ROS2, we utilize python's **pySerial** library, which sends messages over the Linux computer's USB ports. This connects to a USB to UART bridge on the STM32 PCB, running at 921600 baud.

On startup, the *Interface* Node will attempt to connect to the USB device, currently hard-coded as **/dev/ttyUSB0**. If a device is not detected, it will enter *LookBack* mode where we directly connect the robot's current and target positions. This allows us to perform pure software debugging - i.e. running the ROS2 components without the need to connect to a physical STM32. If a device is detected, we will enter normal operation mode and communicate with the STM32.

To allow for better concurrency, we generate two child threads to monitor the USB-Serial interface's input and output respectively. The first thread is in charge of regularly transmitting the current target positions to the STM32, while the second thread monitors incoming data, updating the robot's current state. This allows for both of these operations to operate in parallel, allowing for non-blocking operation.

#### Data Transmit

The *Interface Node* transmits two types of data packages to the STM32: [**p**] for target positions and [**o**] for SEA offsets. Target position data packets consist of 7 floating point values containing the robot's desired joint angles derived from the **/joint_states** ROS2 topic. Before transmitting the data, we must also clamp the target angles, ensuring they do not exceed the robot's joint limits. SEA offset packets consist of 7 floating point values which describe the estimated SEA angle given the current orientation. This data is calculated by the *Gravity Compensation* Node and received through the **/arm_status** ROS2 topic.

When transmitting data to the STM32, the values must first be compressed. Since we have a limited work area, we compress the data per joint into two bytes. This means, along with the indicator character, each packet consists of 15 bytes of data. With the communication speed set to 921000 baud, each message takes approximately $130\mu s$ to transmit.

#### Data Receive

When receiving data from the STM32, all the relevant state data comes in one packet. The information itself is split into multiple parts: [**indicator, joint angle, sea angle, sea offset**]. The *indicator* is a single character indicator indicating we are sending the *joint status*. The joint angle consists of 7 floating point values, denoting the current robot's current orientation in radians. The *sea angle* denotes the current SEA angles (in radians), which allows us to calculate the torque experienced by each joint. The *sea offset* denotes the current expected SEA offset of each joint. This is primarily used for debugging purposes, allowing

us to verify that the STM32 is using the same values that we have sent it.

The information is received in the form of a comma separated string with all the relevant data values, rounded to three decimal places. This allows for ease of use on the python side in translating and storing the data. Each packet consists of 60 bytes of data. With the communication speed set to 921600 baud, each transmission takes approximately $521\mu s$.

## Gravity Compensator

The Gravity compensator is used to calculate the estimated torque due to gravity on each joint depending on the robot's current configuration. This allows us to filter out gravitational forces and focus on unexpected disturbances.

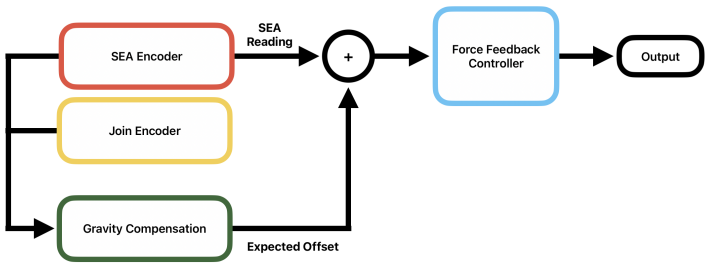
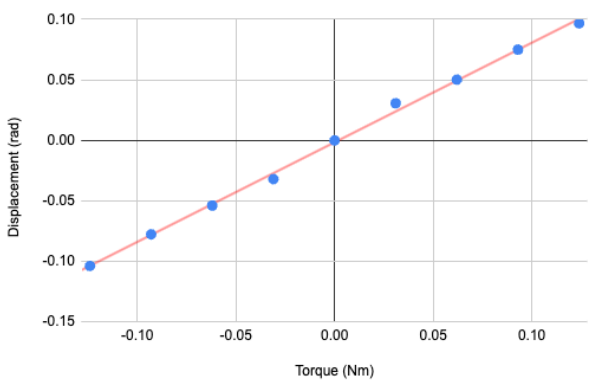


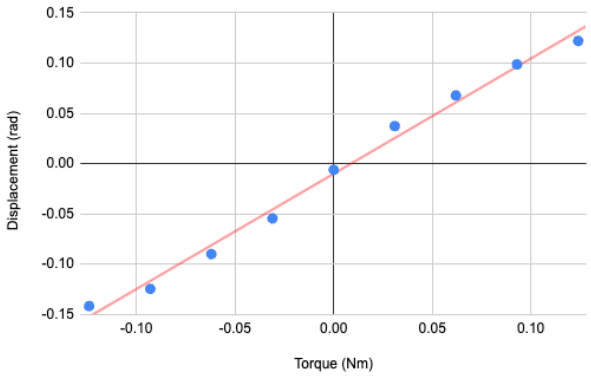

Figure 2.4: Gravity Compensator Block Diagram

Gravity compensation is calculated using the **PyKDL** library. We can use the library to convert the robot arm's URDF file into a kinematic tree, which contains each joint's position, mass, and inertia. Then, we set the gravitational vector as $[0, 0, -9.81]$ and apply it to each of the robot's links, given its current orientation. The *Gravity Compensation* Node receives updated state information from the *STM32 Interface* Node through the **arm_meas** ROS2 topic, meaning the estimated force is continually being updated based on the robots current status. This estimated joint torque is then translated to SEA encoder offset values, which are then published to the **arm_status** ROS2 topic.

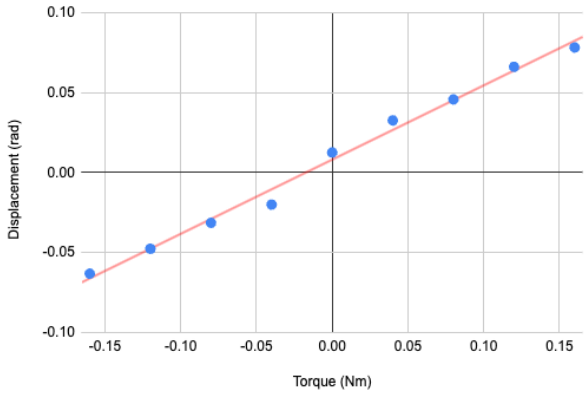
The mapping from joint torques to SEA displacement values was done experimentally through manually applying a set load onto each joint and recording the SEA encoder angle. Separate tests were done for each differential gear joint (Wrist, Shoulder, and Elbow) since each joint used different springs and gear reduction ratios for their SEAs. Each of the calibration plots are shown in Fig. 2.5.




(a) Wrist joint calibration




(b) Elbow joint calibration




(c) Shoulder joint calibration

Figure 2.5: Robot arm calibration plots

We can then model each joint using an affine function, determining the mapping between the estimated torque and the estimated SEA angle. After obtaining the transformations, tests were performed to verify the results, with the model doing a relatively good job of tracking the measured SEA angle. This allows us to ignore gravitational forces when reading the torque along each joint, isolating the relevant data which pertains to the amount of external forces experienced by each joint.

## Visualizer

To better see the current robot state, we also have a built-in visualizer (Fig. 2.6). The 3D visualizer on the left uses rviz to display the robot's current configuration, showing what the robot's current orientation. The live graphs on the right plots the robot's joint angles (blue) and SEA angles (cyan) in tandem with the current SEA estimate (green). This provides an easy way to verify our SEA angle estimates, and see if any of the joints are experiencing unexpected forces.

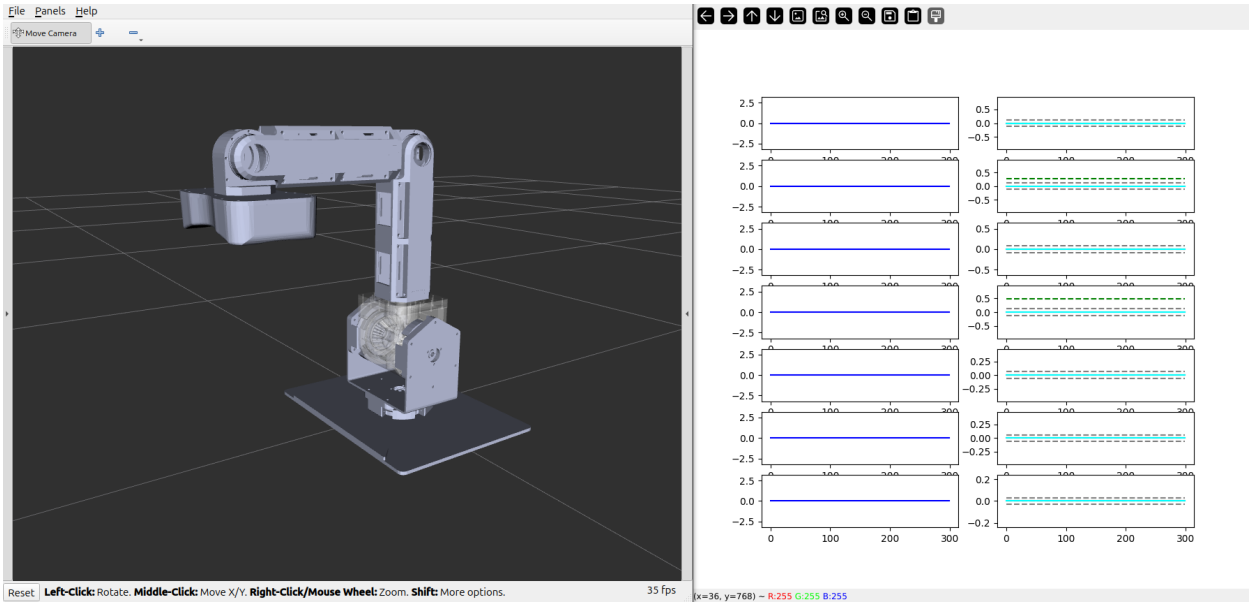



Figure 2.6: Robot state visualizer

# Chapter 3

# Firmware

## 3.1 Framework

While the higher-level tasks (like path planning or inverse kinematics solving) are run on the Linux machine, a STM32 F446RE microcontroller is responsible for the more hardware adjacent tasks. This allows for faster monitoring and control of important low-level tasks, especially in relation to patient safety. The low-level controller's firmware was developed using the Lingua Franca framework [18].

### Lingua Franca

Lingua Franca is an open source embedded system framework developed as a collaboration project between Berkeley and several other research institutions. It was developed with a special focus on stability and determinism, allowing for concurrency without the inherent synchronization issues generated by multi-threaded programs. Lingua Franca primarily acts as a coordination language, organizing and scheduling blocks of conventional C code called reactors. Reactors can be strung together in dependency chains to form complex programs, with Lingua Franca ensuring relations are always deterministic.

Since the robot will be moving around to a patient's head, consistency and predictability are an important step in ensuring safety. Lingua Franca has been proven to ensure safety constraints are met in these situations [19], helping add another level of protection during operation.

### STM32 Flow

Lingua Franca is written using *.lf* files, chaining together blocks of C code and defining the firmware's overall structure. This is then fed into the Lingua Franca Compiler (lfc) which synthesizes everything into C code and triggers the relevant build steps to compile and flash the firmware.

To use Lingua Franca for our project, we needed to add support for the STM32 platform - specifically for our microcontroller, the STM32-F446RE. To allow Lingua Franca to recognize the STM32, we added the necessary *platform.c* and *platform.h* files which implement basic API functions (like enabling/disabling interrupts and interfacing with timers). This was accomplished through the STM32's MX Hardware Abstraction Library (HAL) to interface with IO and hardware devices.

The STM32's cmake build flow also had to be integrated into lfc's cmake generator, which produces all the necessary CMakeLists.txt files. This also includes adding pointers to the STM32 MX HAL library files so that the linker can import relevant binaries. Through running the lfc command, the relevant code is both generated and built, allowing for quick development. Finally, binary flashing was performed using OpenOCD through the ST-link programmer/debugging interface.

## 3.2 Firmware Architecture

The firmware can be found here: [https://github.com/biomimetics/MRIRobot_Firmware.git].

The firmware consists of several reactors, namely: **QDEC**, **HomeController**, **ROSInterface**, **MotorController**, and **USM**. The top level reactor block diagram is shown in Fig. 3.1.

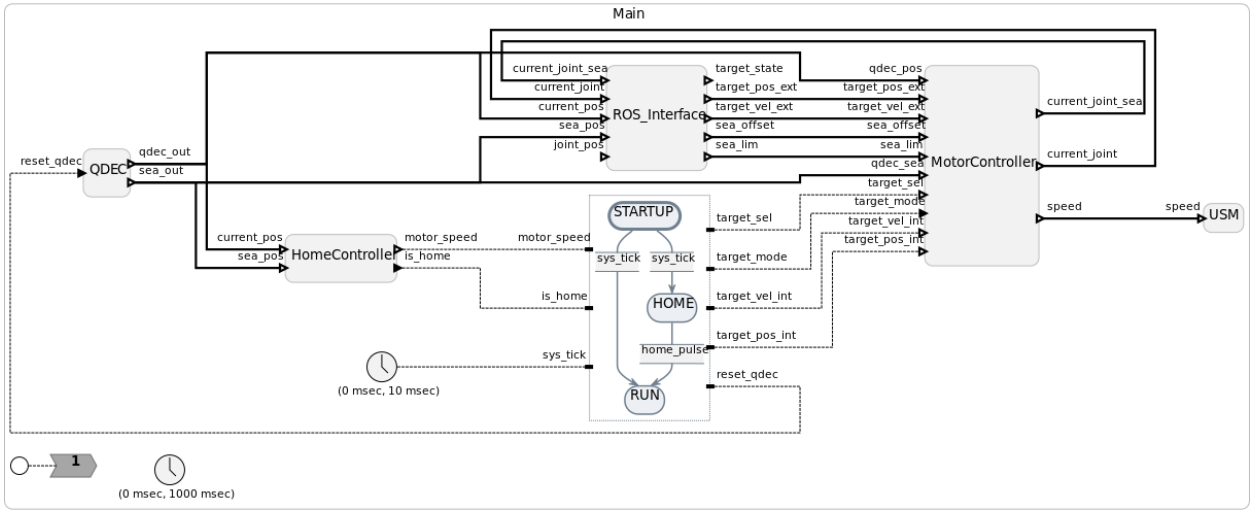



Figure 3.1: Lingua Franca top level block diagram

### QDEC

The **QDEC** reactor acts as the interface between the STM32 and the FPGA, receiving updates on the angles of each encoder (One for each motor and one for each SEA).

The reactor requests new state data from the FPGA every 5 milliseconds. Each data packet consists of 8 bytes, concatenated into one int64 variable denoting the encoder tick count. This is then converted from ticks to radians and then stored on the STM32 as a fp32 value. Conversion from ticks to radians is done through the following:

**Joint Encoder:**

$$encoder_{rad} = encoder_{ticks} \cdot \frac{2\pi}{23040\frac{ticks}{rev}} \cdot \frac{1}{gear\_ratio_{motor}} \tag{3.1}$$

**SEA Encoder:**

$$encoder_{rad} = encoder_{ticks} \cdot \frac{25.4mm}{8000\frac{ticks}{mm}} \cdot \frac{gear\_ratio_{SEA}}{gear\_ratio_{motor}} \tag{3.2}$$

Communication is performed through the STM32s UART controller, running at a speed of 921600 baud. To transfer all 8 bytes describing a single encoder angle, it takes approximately $65\mu s$. Adding together all 14 encoders, this brings the estimated transfer time to approximately $973\mu s$ or $0.973ms$.

To decrease the CPU load, all the UART operations are done through the STM32's built-in DMA. The Direct Memory Access Controller (DMA) is a peripheral which initiates memory operations between the microcontroller's peripherals and/or memory space without CPU interference. If we were to receive data through a blocking statement, we will need to wait $0.973ms$ every $5ms$, meaning the QDEC alone would use approximately 15% of the CPU's total up-time. After requesting new data from the FPGA, the DMA handles receiving the necessary data, storing everything into a local memory address. Receiving data is done completely asynchronously from the main reactor, with minimal impact to the CPU's performance. The **QDEC** reactor can then read the local state data whenever necessary, do the relevant unit conversions, and send it downstream to the other modules.

## HomeController

Since all joints use relative encoders, we must first determine their absolute positioning on startup. The **HomeController** reactor performs this homing sequence to ensure we know what the robot's current state.

During operation, the robot has two possible startup sequences: *Cold Boot* and *Warm Boot*. The Homing sequence is only run on *Cold Boot* where we perform the full initialization sequence. On *Warm Boot* the homing sequence is skipped and we assume the encoders start in their homed position. This is primarily used for debugging and to allow the microcontroller to quickly reset its position estimates without having to go through a lengthy calibration process.

## ROSInterface

The **ROSInterface** reactor acts as the other side of the ROS ⇔ STM32 communication bridge. It handles receiving joint targets from and periodically sends state updates to the ROS computer.

Like with the **QDEC** reactor, the **ROSInterface** also uses the DMA to help minimize CPU load. On the receive end, new target positions and SEA offsets are asynchronously stored in memory, with downstream reactors being updated every $20ms$. On the transmit end, the current joint states are locally storied in memory. They are then transmitted through UART every $50ms$.

## MotorController

The motor controller (Fig. 3.2) consist of multiple individual controllers for each of the robot's joints.

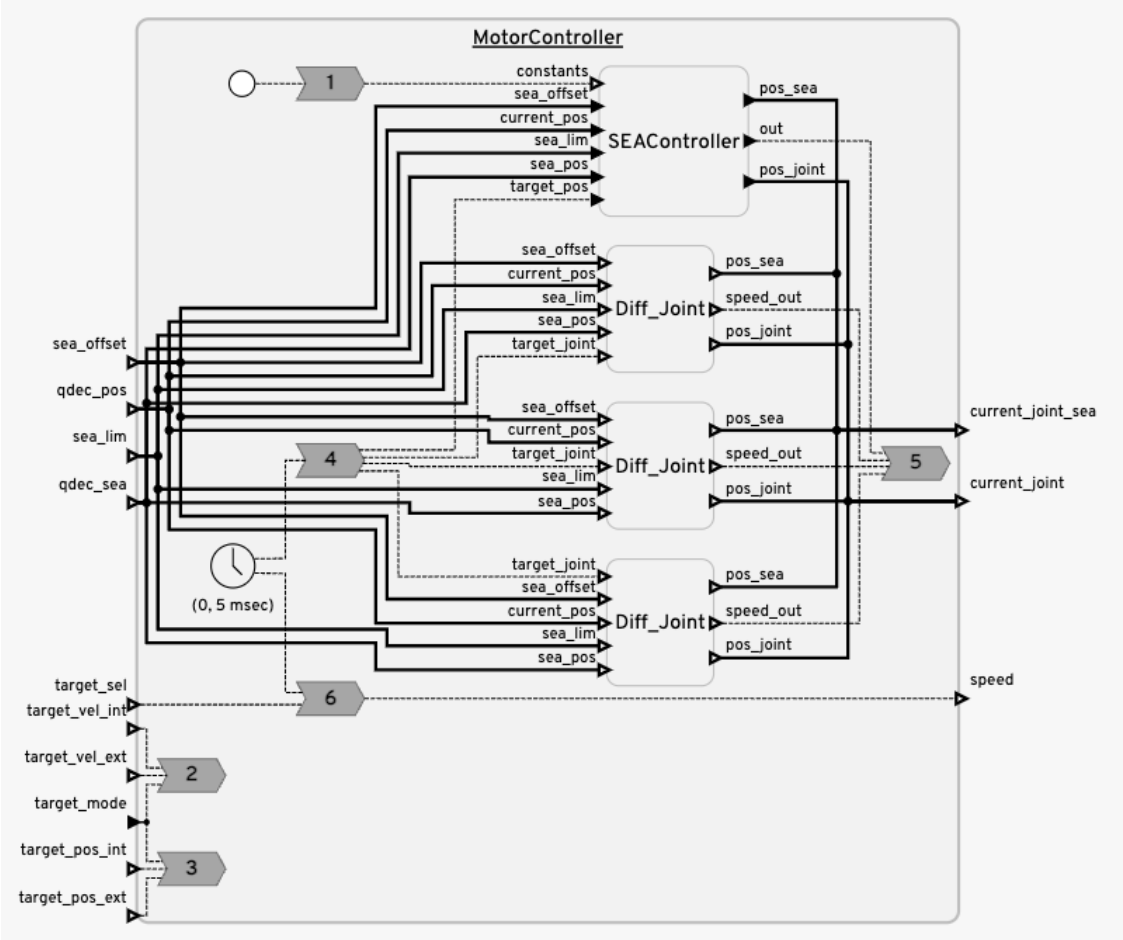



Figure 3.2: Motor Controller Reactor Block Diagram

The robot's Elbow, Wrist, and Shoulder joints are each powered by differential drives, meaning commands in *motor space* (The angles of the motor) must be transformed to *joint space* (The angles of the joints). The bottom rotary axis, however, is directly driven so no transformations are required.

### Diff Joint Controller

The Diff Joint Controller is used to drive each of the differential drives, using the mapping described in Eq. (3.3) to transform angles in *motor space* (MS) measurements to *joint space* (JS).

$$\begin{bmatrix} encoder\_0_{JS} \\ encoder\_1_{JS} \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} encoder\_0_{MS} \\ encoder\_1_{MS} \end{bmatrix} \tag{3.3}$$

For block IO, the target_joint and sea_offset inputs are given in *joint space* while sea_pos and current_pos are given in *motor space* and thus need to be converted. During operation, all control operations are performed in *joint space* coordinates to better follow our high-level control scheme. On the output side, pos_joint and pos_sea are in *joint space*, while the speed_out is in *motor space* and fed directly to the ultrasonic motors.

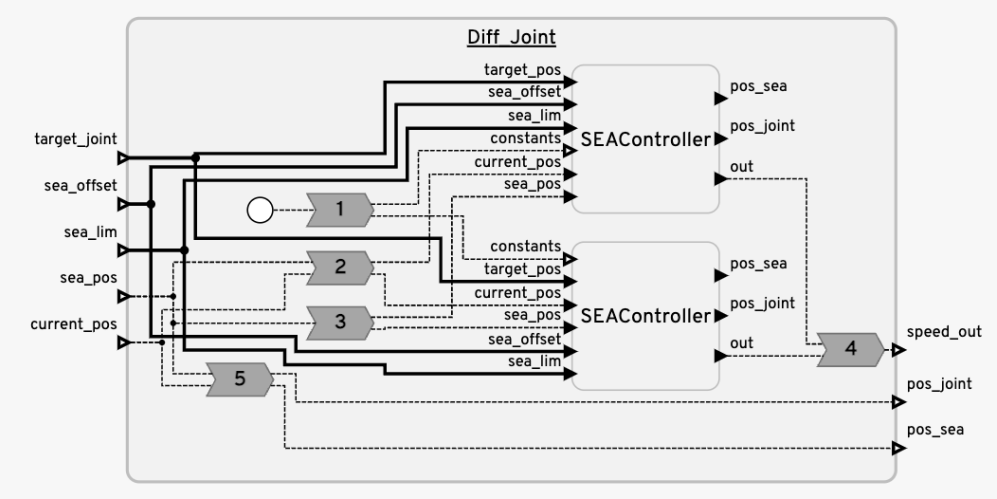



Figure 3.3: Diff Joint Controller Reactor Block Diagram

The reactor itself (Fig. 3.3) consists of two **SEAController** reactors, one for each of the two axes - pitch and roll. Each joint is operated independently for both position control and force feedback.

### SEA Controller

Each joint is controlled by a **SEA Controller** reactor which consists of two modules: a **PIDController** and a **FFBController** placed in parallel.

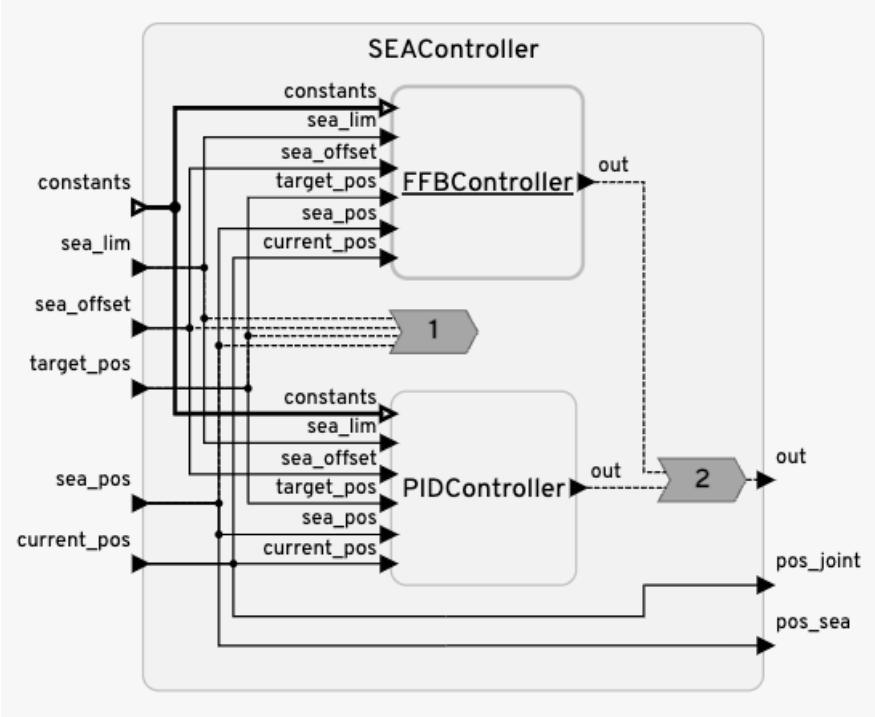



Figure 3.4: SEA Controller Reactor Block Diagram

The SEA controller (Fig. 3.4) has an internal finite state machine of two states. Normally, the joint will operate in *PID mode*,acting as a standard PID position controller. However, when the SEA angle is above a certain threshold - indicating an unexpectedly large torque - we switch to *FFB mode*. In *FFB mode*, the goal is to zero the SEA's angle, simulating full back-drivability on the arm's joints. We will stay in this state until we receive a new target position, where we will transition back into the *PID mode* state.

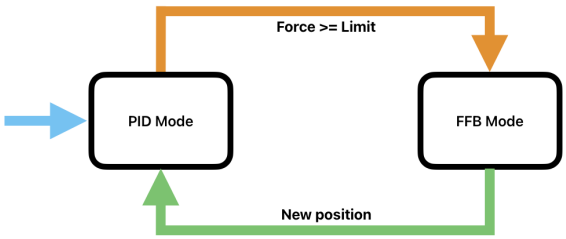



Figure 3.5: SEA Controller Finite State Machine

For gravity compensation, we provide an offset to the SEA encoder angle as a way to indicate an expected torque. This allows the robot ignore any gravitational forces and only trigger the SEA FSM based on external forces.

### USM

The **USM** reactor acts as the actual interface for all the robot's ultrasonic motors. The reactor takes in floating point values describing each of the motors desired speeds, converting them to a control input signal. The motors themselves consist of a relatively basic control scheme: PWM for speed and two pins for direction and enable respectively.

Before directly setting the output values, we also clamp our speed values, imposing a maximum speed on each of the robot's joints. Since we are operating within an MRI bore and need to be wary of eddy currents, the robot must move relatively slowly. We also implement a motor start/stop switch in software so that there is a way to quickly disable the motors if needed.

# Chapter 4

# Electronics

## 4.1 Overview

The robot arm's electronics are controlled by a STM32 microcontroller housed on a primary printed circuit board (PCB). Signals are then broken out to the Ultrasonic motor controllers (provided by Tekceleo) and a number of intermediary PCBs which help with rerouting various signals. All the electronics are designed to sit outside the MRI scanning room, with cables being funneled through a hole in the wall. A block diagram is shown in Fig. 4.1.

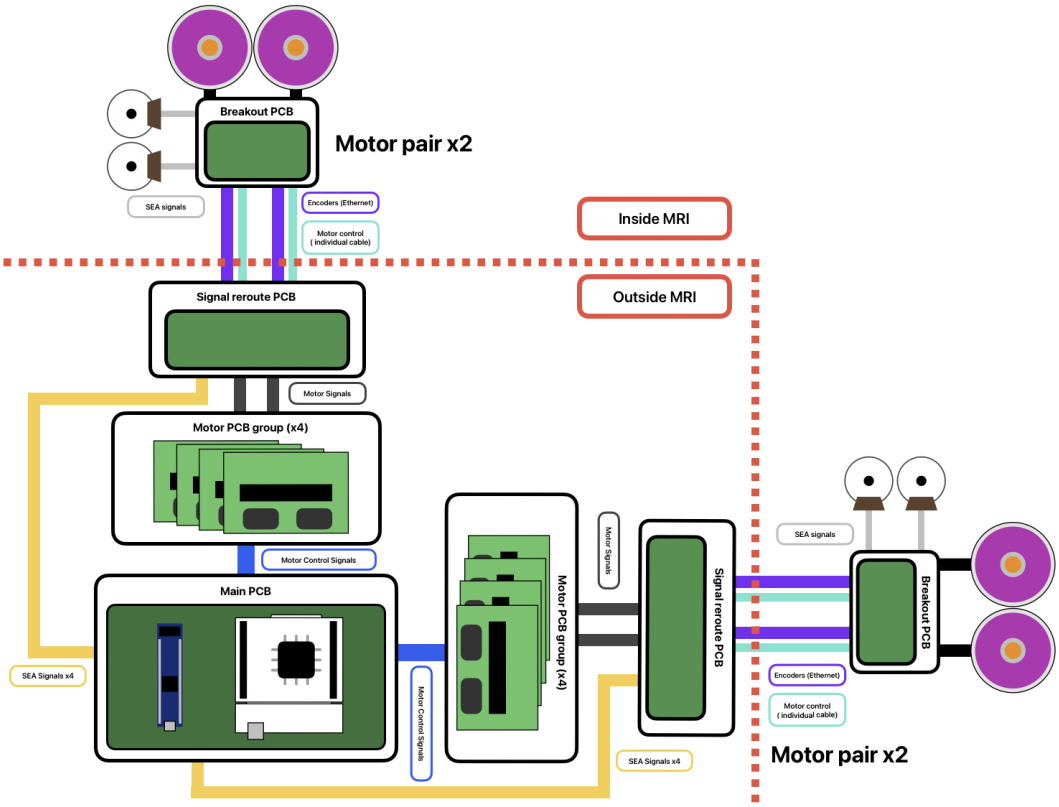



Figure 4.1: Electronics block diagram

## 4.2 Circuit Boards

The robot's PCBs can be found here: [https://github.com/biomimetics/MRIRobot_PCB.git].

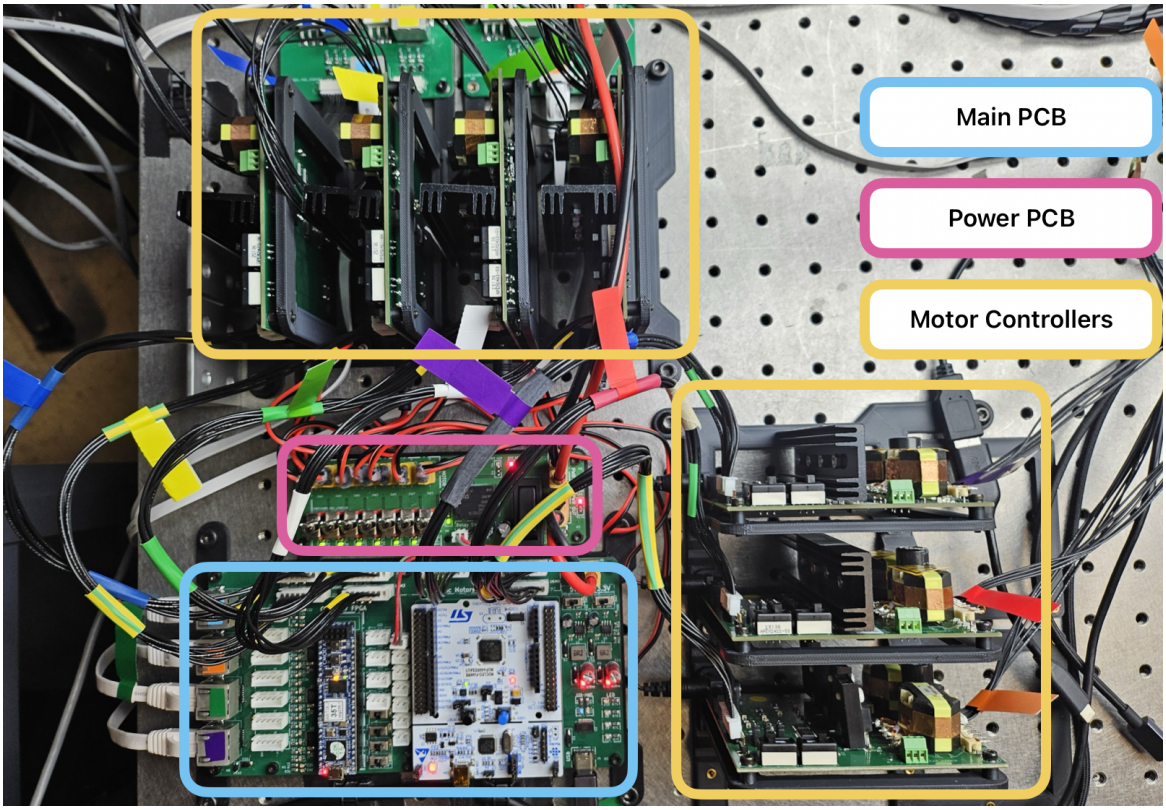
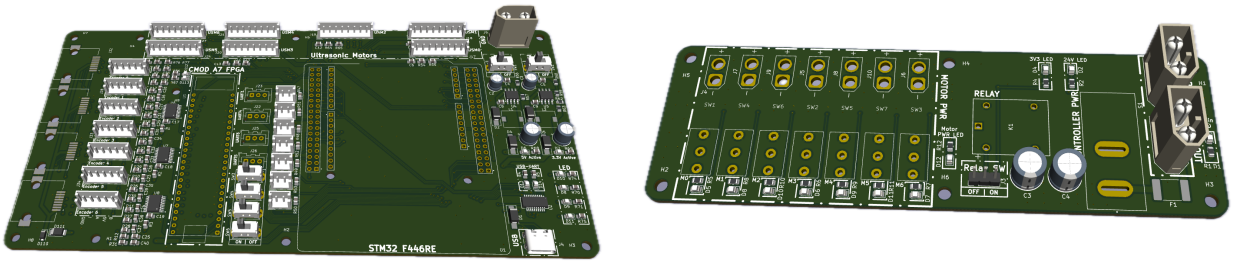



Figure 4.2: Robot control PCBs

### Main PCB

The main PCB is broken into two components: compute and power. The main compute PCB is a 4-layer board containing our low-level microcontroller (The STM32 F446RE) and FPGA co-processor (Xilinx Artix-7 35T). The board contains the necessary IO breakouts to control each of the 7 Ultrasonic motors, and read data from the SEA encoders. It also contains its own power step-down, taking in 24V as the input, and converting it to the necessary 5V and 3.3V to power the board's components. Since the SEA encoders operate at 5V, we also include level shifters so we do not damage the 3.3V microcontroller.

We also have a secondary power distribution PCB which is a two-layer board manufactured using 2oz copper, primarily used to provide power to the ultrasonic motor controllers. Each ultrasonic motor can draw up to 3A at 24V, meaning the power board must be capable of handling a max current of over 21A. All power to the motor controllers is routed through a relay, allowing our microcontroller to act as a dead-mans-switch for the system. This ensures that the motors can only be powered when the STM32 is active, adding an additional level of safety.

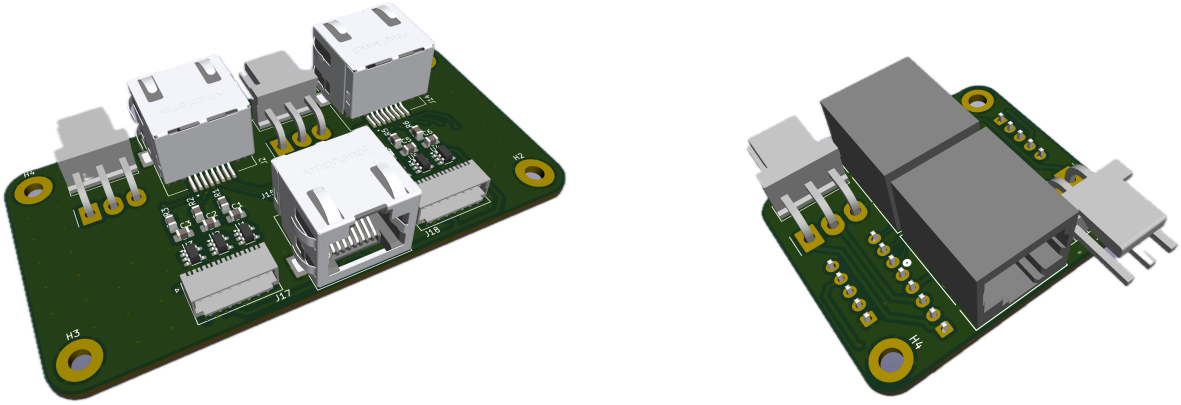



(a) Main compute PCB (b) Main power PCB

Figure 4.3: Main PCBs

### Signal Reroute PCB

To minimize the number of cables going from the robot to the control room, we designed a number of signal reroute PCBs. These worked to consolidate the signals into two wires per motor: one 3-pin MTA-156 header pin containing the ultrasonic motor's control signals, as well as an Ethernet cable containing the encoder signals from both the motor and SEA. Connector selection was especially important for the motor control signals, since they operate at 220V, meaning we had to ensure everything was high-voltage compliant. Between all 7 motors, we have a total of 14 cables going from the control room into the MRI room.



(a) Controller side reroute PCB (b) Robot arm side reroute PCB

Figure 4.4: Signal Reroute PCBs

## 4.3 FPGA

The robot's FPGA RTL can be found here: [https://github.com/biomimetics/MRIRobot_FPGA.git].

To calculate the joint and SEA displacements, we need 2 rotary encoders per axis, totaling 14 encoders for the entire robot. These encoders work by reading offset two square signals - A and B - with the sequence of offset rising/falling edges indicating the direction and

displacement. Performing those calculations on a CPU can be quite costly, with most implementations relying on either polling (in which case you can lose steps) or interrupts (which can lead to a decrease in CPU performance as the ISR is constantly firing in the background).

We use a FPGA co-processor to perform these calculations in hardware, allowing for faster speed and parallel processing. We sample at 4 points along a single duty cycle (A_rise, B_rise, A_fall, B_fall) which allows us to achieve 23040 steps per rotation for the ultrasonic motor and 89759 steps per rotation for the SEA encoder. With a FPGA clock speed of 125MHz, we can operate at a maximum speed of 34088 rad/s on the ultrasonic motor and 8750 rad/s on the SEA encoders - Both well beyond our requirements.

As seen in Fig. 4.5, the architecture of the FPGA consists of 14 QDEC block modules (7 for the SEAs and 7 for the ultrasonic motors), an arbiter, Serdes module, and UART transmitter/receiver. All IO inputs are passed through a synchronizer to help align IO signals to the FPGA's internal clock, ensuing data is properly timed and helping remove IO capture errors.

Once a request has been received from the UART module, the FPGA will load the desired QDEC's data into the FIFO where it will be fed into the serdes module. Since the encoder ticks are storied as int64 and we can only transmit 8 bits at once through UART, the tick count must be serialized into 8 packets. These packets are then transmitted one at a time through the UART port. Along with getting encoder ticks, there are also commands for resetting encoder values, as well as disabling the QDEC modules which stops the counters from incrementing (this is used for when the MRI is scanning and there could be noise on the encoder wires).

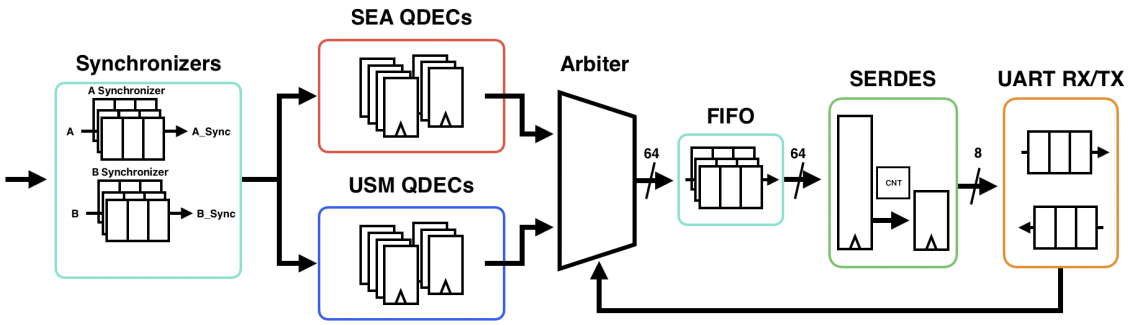



Figure 4.5: FPGA top-level RTL block diagram

As seen in Fig. 4.6, the QDEC modules consist of an edge detector, control logic, and counter. For the SEA QDEC modules, since we don't know if the springs are pre-loaded on startup, we take advantage of the encoder's index pin to get absolute positioning. The counter is reset when the index signal is asserted, with a manual offset being applied to zero each of the SEA encoder angles.

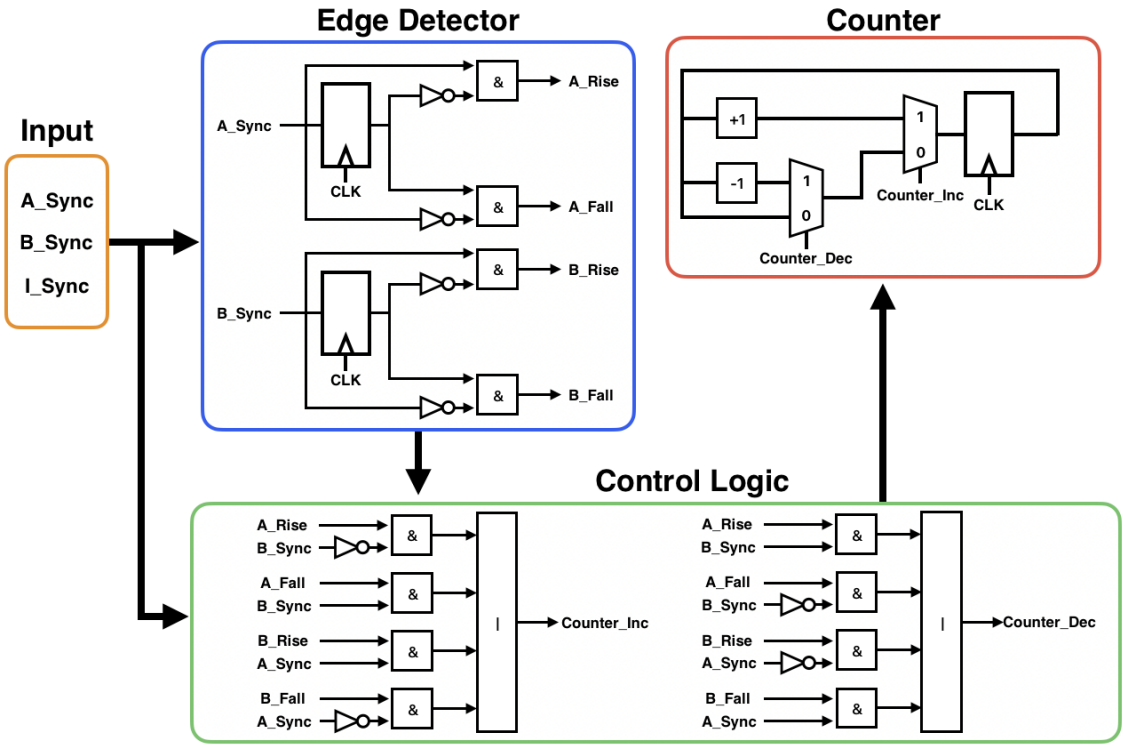



Figure 4.6: FPGA QDEC module block diagram

## 4.4 Cables and MRI Compatibility

By default, Tekceleo uses Pico SPOX 87438 connectors to connect their ultrasonic motors to their motor controllers. However, these cables are not shielded very well and have exposed metal contacts, which is dangerous. These cables combine the motor's control signals with the encoder signals coming from the ultrasonic motor's built-in encoders.

| Pin | Name | Direction | Voltage |
|---|---|---|---|
| 1 | Sin Wave | Output | 220V |
| 2 | GND | - | - |
| 3 | Cos Wave | Output | 220V |
| 4 | Encoder Power | Output | 5V |
| 5 | GND | - | - |
| 6 | Encoder A | Input | 3.3V |
| 7 | Encoder B | Input | 3.3V |
| 8 | Encoder Index | Input | 3.3V |

Table 4.1: Ultrasonic motor wire signals

Instead, we decided to design our own cable interface between the control PCB and the ultrasonic motors, allowing us to better control the shielding and signal grouping. Each motor consists of two cables:

1. A **3-pin MTA-156** used to connect the motor control signals ***[sin, cos, gnd]***.
2. A **Ethernet** containing all the signals for both the motor's and SEA's encoders.

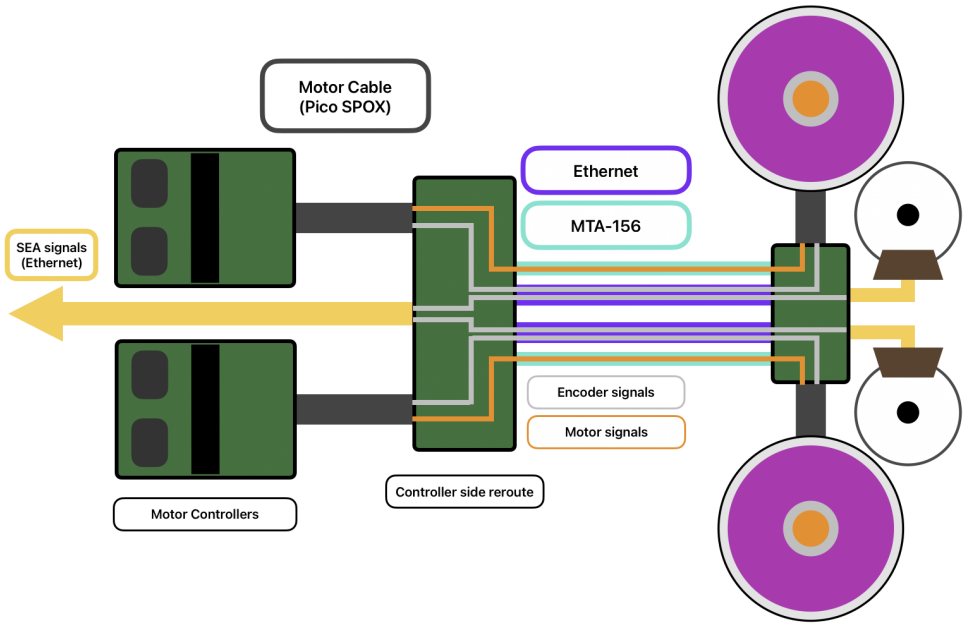



Figure 4.7: Cable routing diagram

The motor control wires are connected through a 3-pin shielded cable. These signals are also purposefully separated from our encoder wires to minimize induced interference, especially since the ultrasonic motors operate at 220V. However, since the motors use such high voltages, they are less prone to induced noise from the MRI machine.

The encoder signals, on the other hand, are routed through an 8-pin shielded Cat-7 Ethernet cable. The cable is shielded to minimize external interference which is especially important since these signals operate at only 3.3V. These signals also go through a number of filters before actually reaching the FPGA.

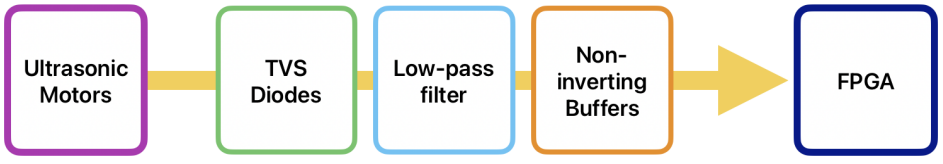



Figure 4.8: Encoder filter block diagram

First, we use TVS diodes as voltage protection to protect our electronics. The MRI can induce significant voltage spikes the control wires, leading to instances where the microcontroller has become damaged. The TVS diodes work to shunt any high transients, ensuring that everything downstream remains safe. Then, we use a set of low-pass filters to remove high-frequency interference. The encoder output has a maximum frequency in the 10kHz range, while the MRI machine induces noise in the 100MHz range. Thus, we place the filter poles at around 1.6MHz to attenuate MRI noise. Finally, we noticed some GND noise issues on the ultrasonic motor's encoders which could be problematic. To ensure proper separation, we added a set of non-inverting buffers to help better separate the signals.

# Chapter 5

# Integration

## 5.1 Results

### MRI Cable Testing

First, we tested the performance of our custom cable setup. To verify this, we had the motor oscillate between two positions $[0, 0.5]$ graphing the position over time. Based on our testing (Fig. 5.1), the two movement profiles look almost identical, meaning that changing the cable made no significant impact on the motor's movements.

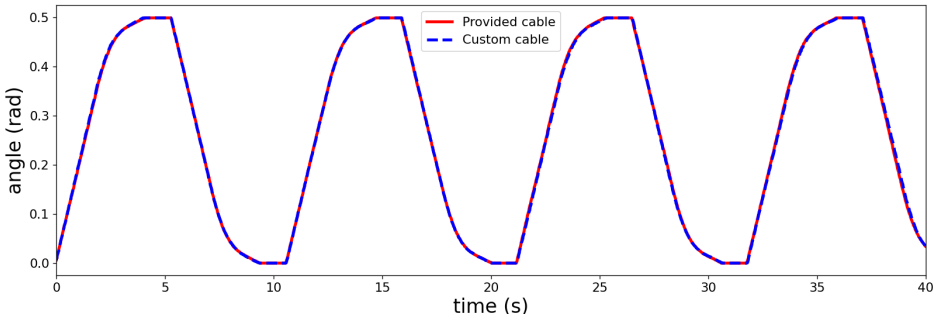



Figure 5.1: Encoder angle over time from using the provided cable *(red)* and our custom cable *(blue)*

Then, we tested the custom cables inside the MRI machine to see its resilience to MRI induced noise. This involved monitoring the motor's SEA readings inside the MRI while performing a scan. Our main concern was the MRI's changing magnetic fields causing the voltages levels in the encoder cables to oscillate, creating phantom steps.

First, we tested the stationary motor inside the MRI machine. The goal is to ensure the encoder readings to not change due to interference, causing the encoders to lose steps. As seen in Fig. 5.2, the motor shielding combined with the filters resisted the induced magnetic noise, as evidenced by the encoder reading remaining stationary across multiple MRI scans.

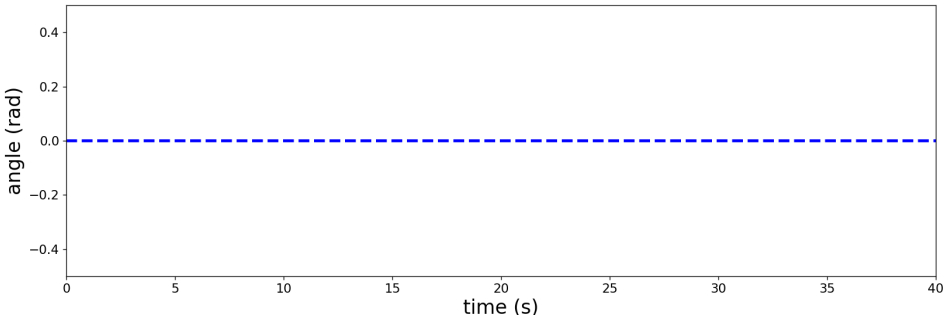



Figure 5.2: Encoder angle over time while inside a scanning MRI machine

Then, we tested moving the motor inside the MRI machine. As seen in Fig. 5.3, we also observe that the cable shielding is able to protect our controller and encoder signals across multiple MRI tests. This means we could be able to operate the robot arm within the MRI bore during a scan.

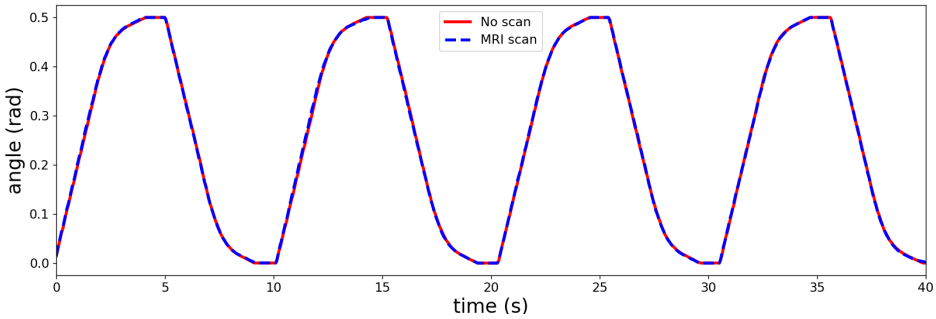



Figure 5.3: Encoder angle over time while the MRI is scanning*(blue)* and idle*(red)*

We also performed tests to verify the motor's velocity control. This involved alternating the desired velocity input between 0 and a set value, creating a square wave. Once again, we observe promising results when comparing the results with and without the MRI scanning Fig. 5.4.

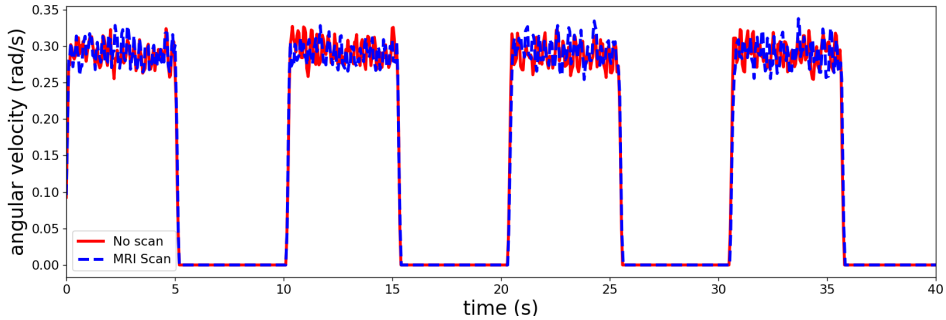



Figure 5.4: Encoder velocity over time while the MRI is scanning*(blue)* and idle*(red)*

However, one issue we did encounter was the fact that the cables could not be completely shielded, as there is a small sliver of wire which adapts the shielded cables to the ultrasonic motor. This creates a gap in which the MRI signals can still interfere with the encoder counts. A demonstration of a stationary encoder is seen in Fig. 5.5 where it is pulled along the bore of a scanning MRI machine. There is a small segment (indicated by the dotted lines) where phantom steps are observed.

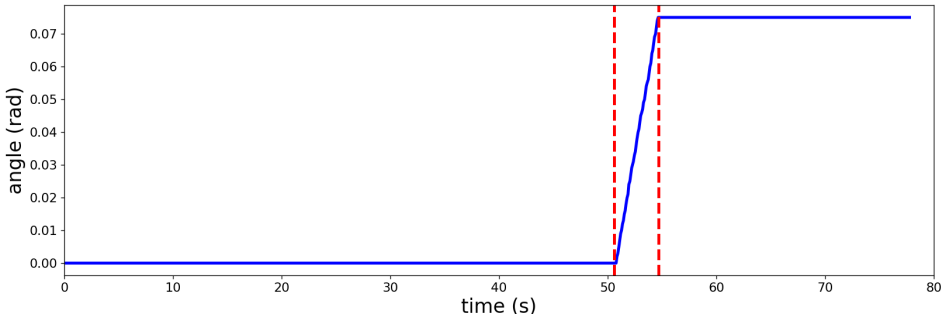



Figure 5.5: Encoder angle while being pulled through a scanning MRI bore

This noise leakage also interferes with the motor's movement. In Fig. 5.6, the motor was pulled through the MRI bore again, performing the same movement profile as above. Due to the MRI noise, the ultrasonic motor's built controller is confused by the noise encoder wires, interfering with the motor's movement.

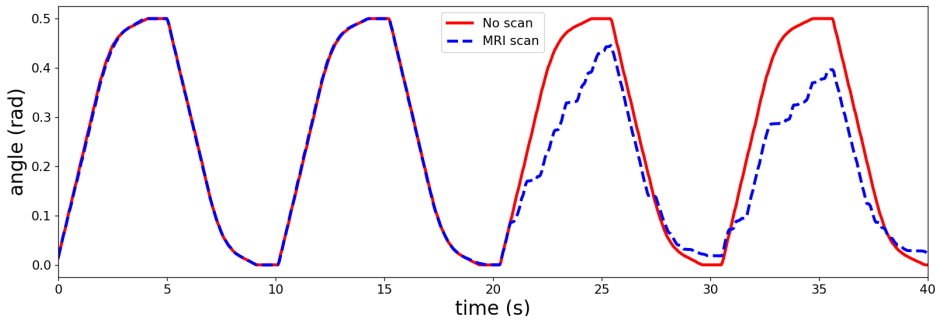



Figure 5.6: Encoder angle over time while the MRI is scanning*(blue)* and idle*(red)* (Only the section of movement where the errors appear)

A similar outcome can also be seen when performing the velocity control tests (Fig. 5.7).

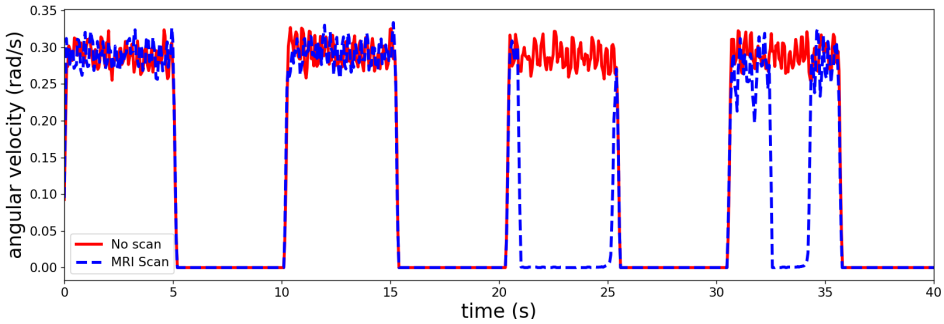



Figure 5.7: Encoder velocity over time while the MRI is scanning*(blue)* and idle*(red)* Only the section of movement where the errors appear)

However, with better shielding along the entire cable, our issues can be attenuated. As seen in Fig. 5.8 even adding basic copper shielding around the exposed areas can greatly improve tracking.

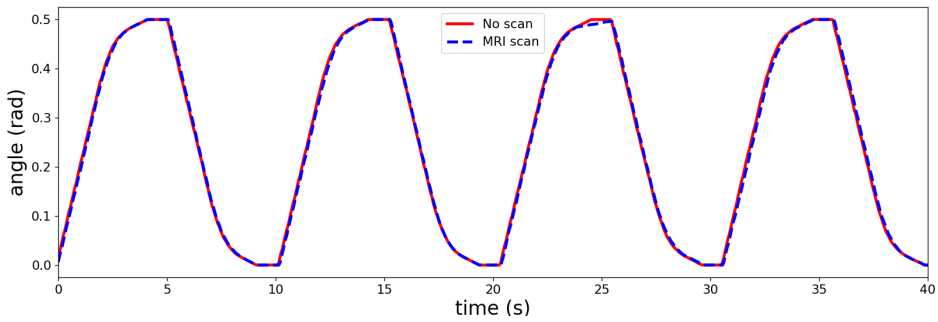



Figure 5.8: Encoder angle over time while the MRI is scanning*(blue)* and idle*(red)*

Thus, we are able to demonstrate that the shielded cables and filters work to remove MRI induced interference where they exist. However, when the shielding is not present, we still see evidence of encoder noise. Nevertheless, our results indicate that it is viable for us to move the robot while the MRI is scanning, allowing us to better utilize the MRI itself as a way of localization [16].

## Robot Homing

On boot-up, the robot performs its initial calibration sequence to home and get the absolute angle of each of its joints. This involves moving each joint towards its defined limits, using the SEA's spring deflection as a way to determining the bounds (and thus the angle) of each of the joints.

A video of the homing sequence: [https://youtube.com/shorts/rpanKMerBhk].

## Zero-G Mode

Zero-G mode works by using the gravity compensator to filter out the gravitational forces on the robots joints. This allows us to isolate and only respond to external forces. In Zero-G mode, the user is able to move the robot to whatever position they desire with minimal force. Once the robot has been moved to a desired pose, the arm will retain its current orientation without drooping due to gravity.

A video of the arm in Zero-G mode: [https://youtube.com/shorts/kUzvczj3raE].

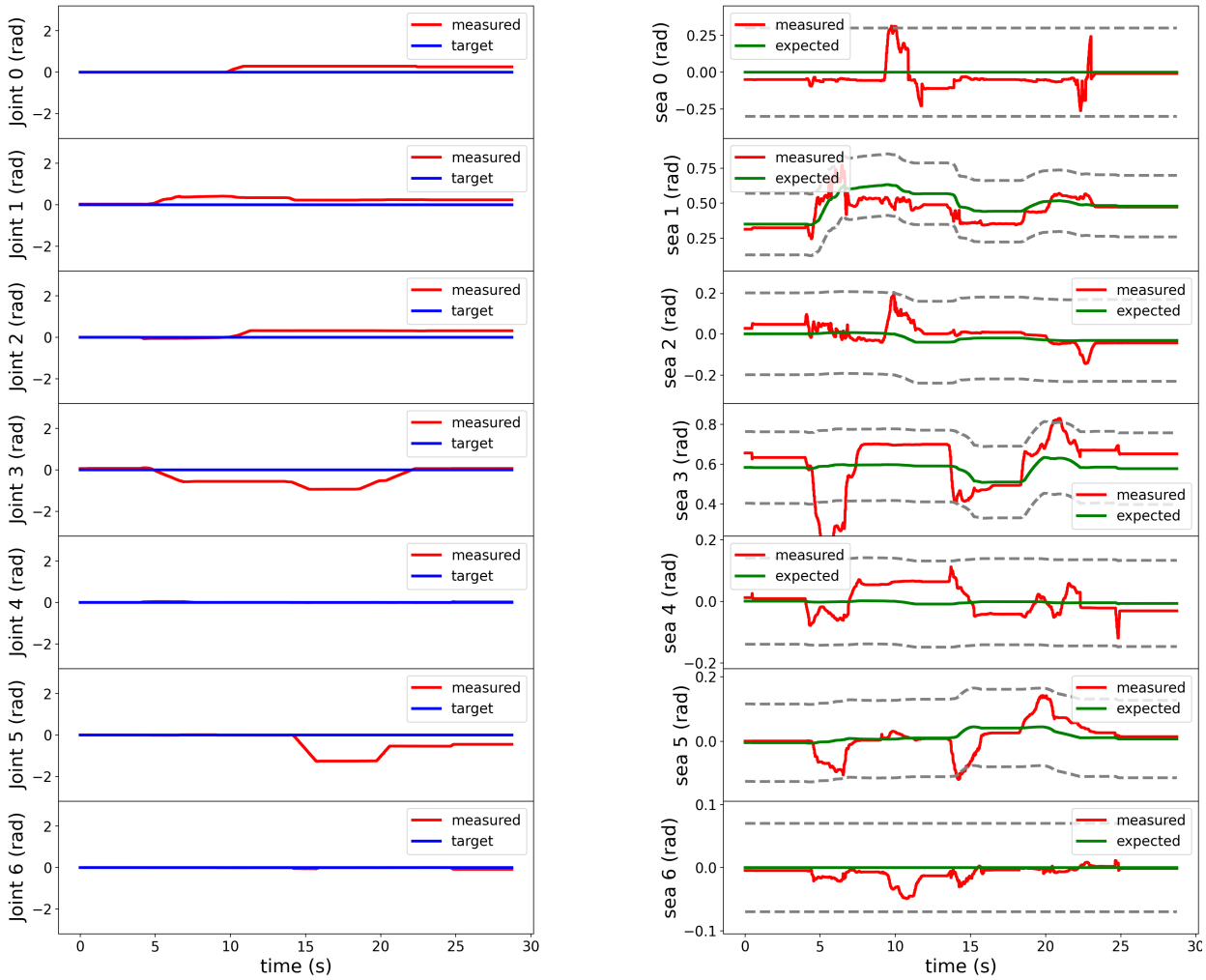



Figure 5.9: data from Zero-G manual movement. Joint angle (**left**) with measured angle (*red*), target angle (*blue*) and SEA data (**right**) with measured displacement (*red*), expected displacement(*green*), error bounds (*grey*).

## Force Sensing

Using the gravity compensator to isolate external forces, we are able to isolate external forces acting on the arm. Therefore, any spikes in observed torque are due to the arm coming into

contact with an unexpected object, telling the robot that it need to stop moving.

Below, Fig. 5.10 shows the robot arm coming into contact with a vertical wall. When a spike in the joint torque is detected, the robot stops in place.

A video of the test: [https://youtube.com/shorts/9NbPLNKhEzY].

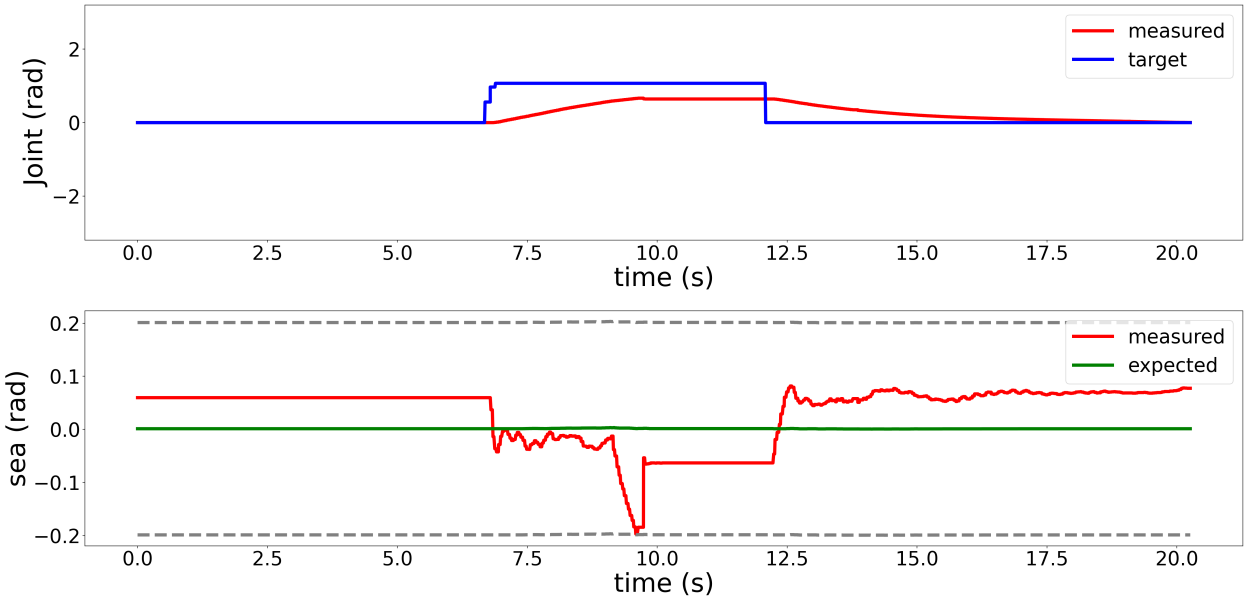



Figure 5.10: Data from arm hitting wall. Joint angle (**top**) with measured angle (*red*), target angle (*blue*) and SEA data (**bottom**) with measured displacement (*red*), expected displacement (*green*), error bounds (*grey*).

Similarly, Fig. 5.11 shows the robot arm coming into contact with the table. When a spike in the joint torque is detected, the robot stops in place.

A video of the test: [https://youtube.com/shorts/sXyTCPKkxF8].

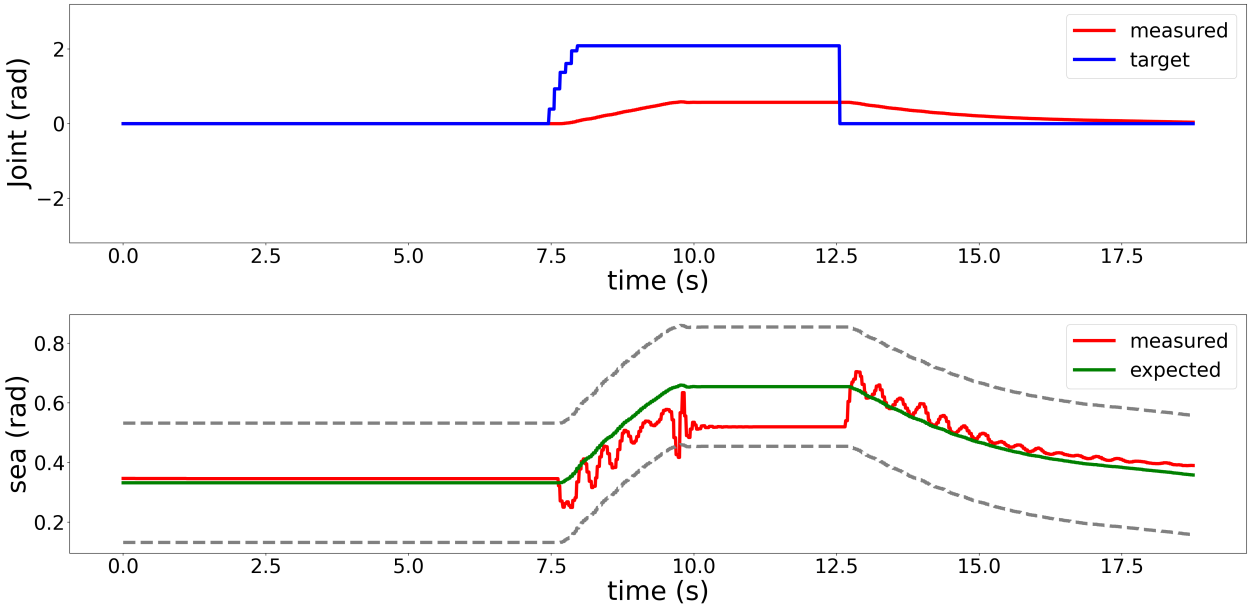



Figure 5.11: Data from arm hitting table. Joint angle (**top**) with measured angle (*red*), target angle (*blue*) and SEA data (**bottom**) with measured displacement (*red*), expected displacement (*green*), error bounds (*grey*).

## Position control

With the rest of the pipeline completed, the robot is able to operate with full ROS2-based position control. In this case, desired joint angles are fed from a ROS2 controller node to the *STM32 Bridge* Node. Then, the bridge forwards that information to the STM32 which uses the PID joint controller to move the arm. The Robot's current state is also sent back from the STM32 into ROS2 so that the user can visualize current joint angles.

To also test the controller's robustness, we also tested across multiple end-effector weights to make sure it would automatically adjust estimations based on the expected weight. Since dynamics are derived from the robot's URDF, a user only needs to update the end-effector mass descriptor to change the robot's gravity compensation estimates.

### Position control 0.9kg weight

The first movement test was performed with a end-effector mass of 0.9kg or 2lb. Below, Fig. 5.12 shows the joint space data.

A video of the test: [https://youtube.com/shorts/2tNHJcir_Ok].

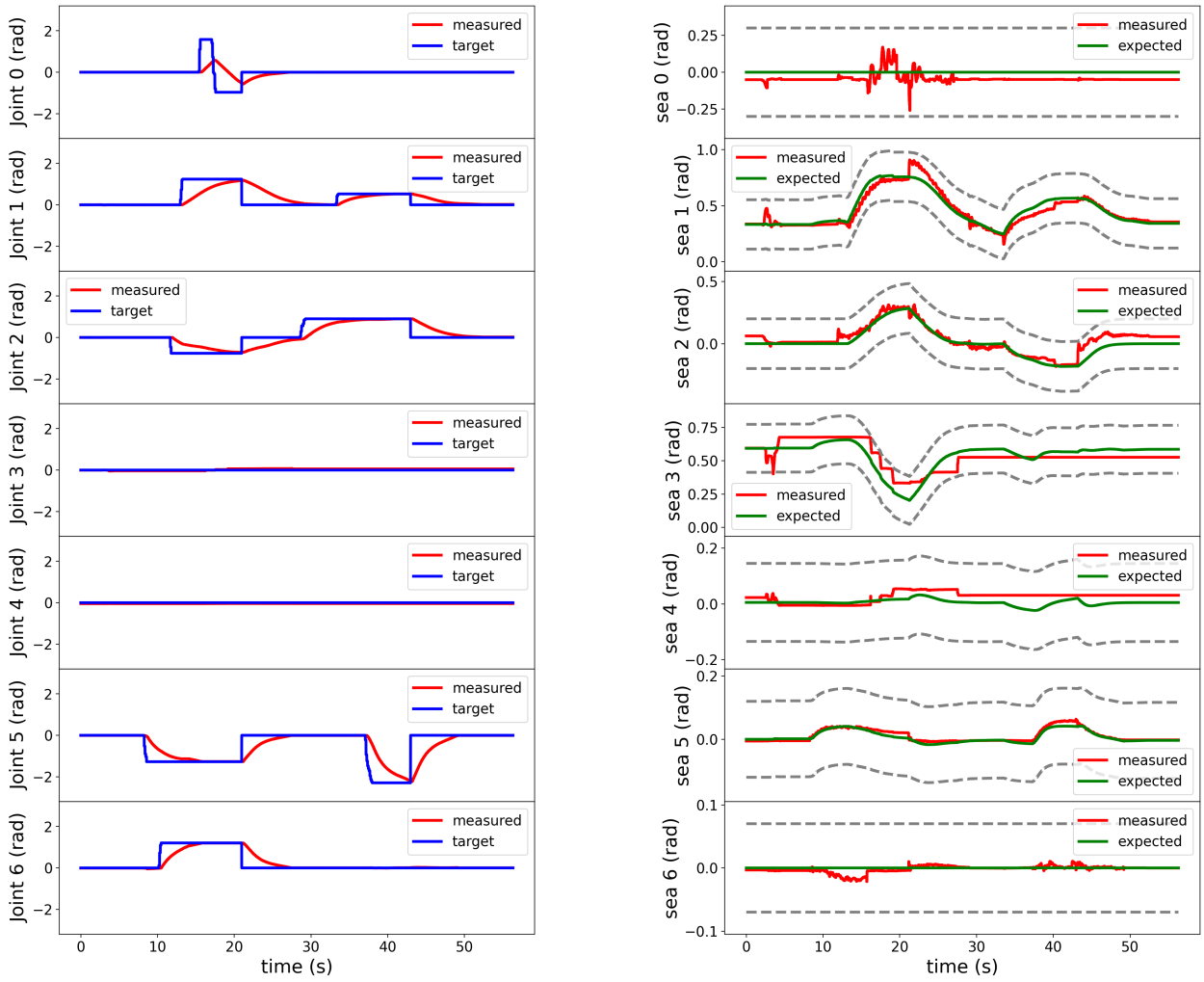



Figure 5.12: Data from position control with 0.9kg end-effector. Joint angle (**left**) with measured angle (*red*), target angle (*blue*) and SEA data (**right**) with measured displacement (*red*), expected displacement (*green*), error bounds (*grey*).

We also performed a test to see the accuracy of the PID control scheme based on how close it is able to move its end-effector to a desired location. Fig. 5.13 shows the end-effector workspace positions, and Fig. 5.14 shows the workspace error.

A video of the test: [https://youtube.com/shorts/pbG8SM9nE1c].

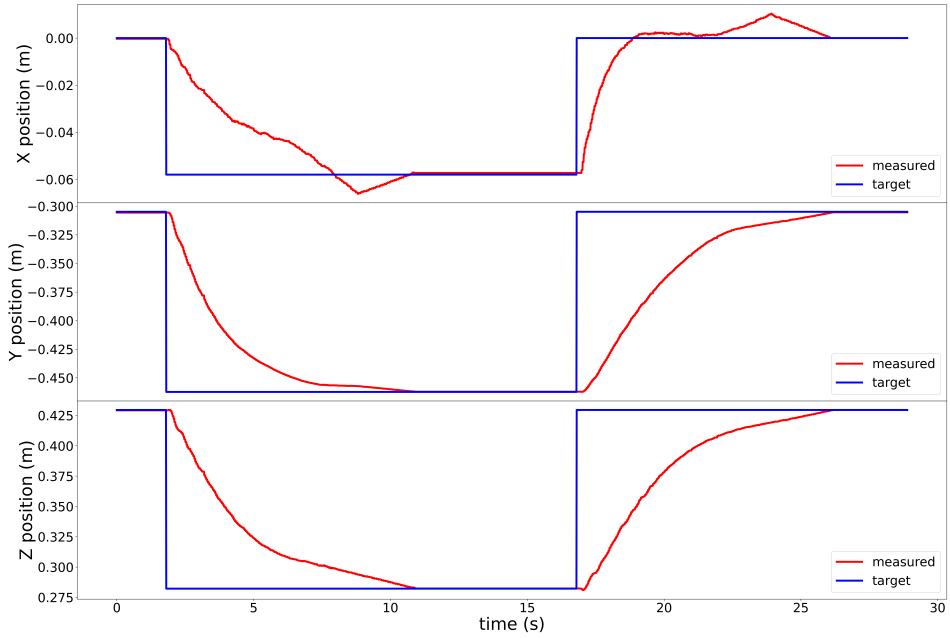



Figure 5.13: Workspace data from position control with 0.9kg end-effector. Measured joint angle (*red*), target angle (*blue*).

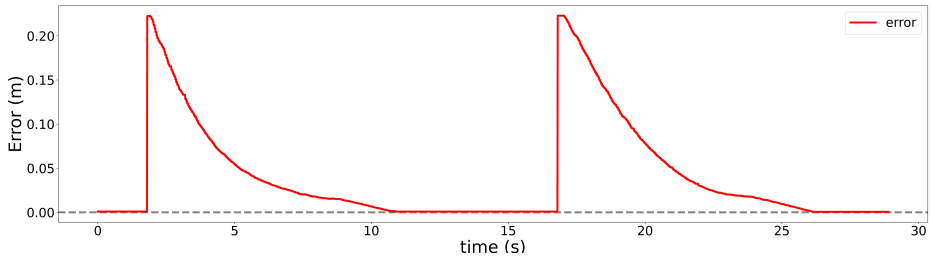



Figure 5.14: Workspace error from position control with 0.9kg end-effector.

Table 5.1 shows the estimated end-effector position errors based on the desired and actual positions taken from joint encoders.

| Measurement | Trial 1 | Trial 2 | Trial 3 | Trial 4 | Trial 5 | Mean | STD |
|---|---|---|---|---|---|---|---|
| Target error (cm) | 0.100 | 0.070 | 0.0456 | 0.106 | 0.209 | 0.106 | 0.0624 |
| Return error (cm) | 0.057 | 0.104 | 0.112 | 0.069 | 0.128 | 0.094 | 0.030 |

Table 5.1: Workspace position errors with 0.9kg end-effector.

### Position control 1.3kg weight

The second movement test was performed with a end-effector mass of 1.3kg or 3lb. Below, Fig. 5.15 shows the joint space data.

A video of the test: [https://youtube.com/shorts/aOf9k9tYXgw].

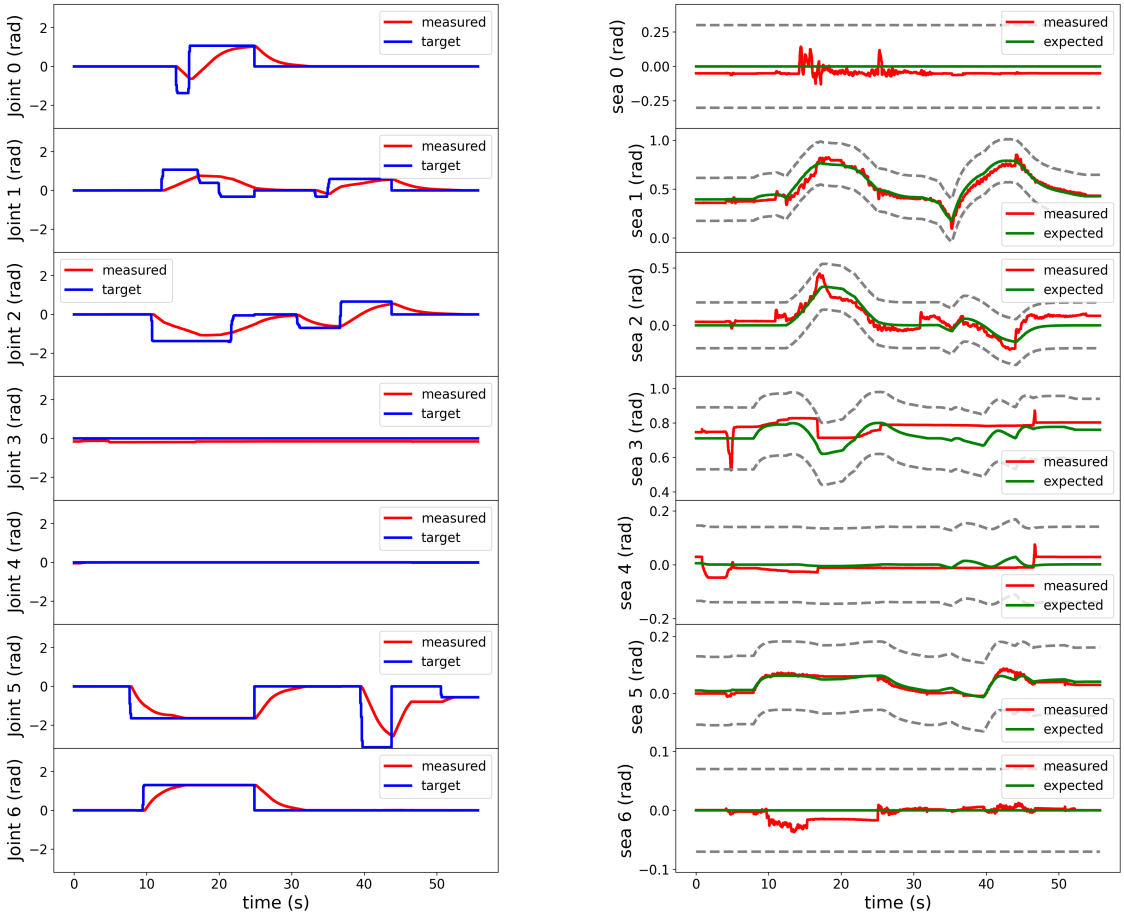



Figure 5.15: Data from position control with 1.3kg end-effector. Joint angle (**left**) with measured angle (*red*), target angle (*blue*) and SEA data (**right**) with measured displacement (*red*), expected displacement(*green*), error bounds (*grey*).

We also performed a test to see the accuracy of the PID control scheme based on how close it is able to move its end-effector to a desired location. Fig. 5.16 shows the end-effector workspace positions, and Fig. 5.17 shows the workspace error.

A video of the test: [https://youtube.com/shorts/ILhWXRuAdxo].

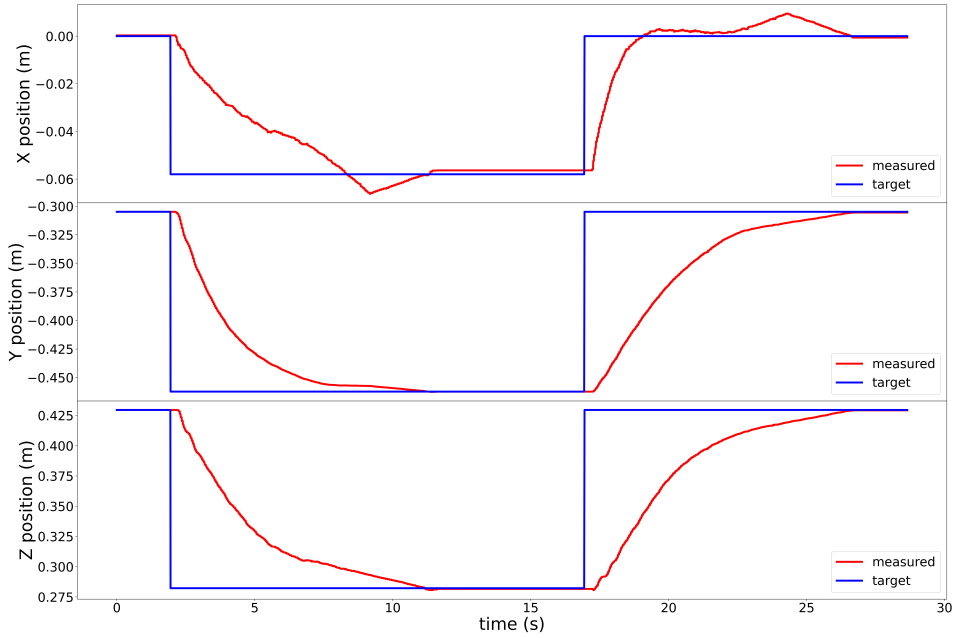



Figure 5.16: Workspace data from position control with 1.3kg end-effector. Measured joint angle (*red*), target angle (*blue*).

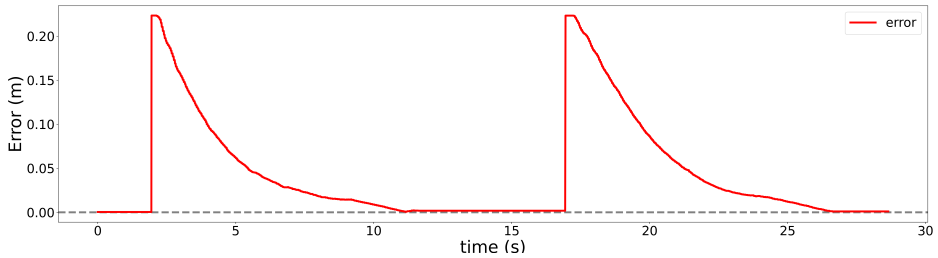



Figure 5.17: Workspace error from position control with 1.3kg end-effector.

Table 5.2 shows the estimated end-effector position errors based on the desired and actual positions taken from joint encoders.

| Measurement | Trial 1 | Trial 2 | Trial 3 | Trial 4 | Trial 5 | Mean | STD |
|---|---|---|---|---|---|---|---|
| Target error (cm) | 0.326 | 0.097 | 0.129 | 0.311 | 0.129 | 0.1984 | 0.111 |
| Return error (cm) | 0.087 | 0.069 | 0.091 | 0.026 | 0.084 | 0.0714 | 0.0267 |

Table 5.2: Workspace position errors with 1.3kg end-effector.

# Chapter 6

# Conclusion

In this report, we described the steps taken to integrate and test a custom-built MRI compatible robot arm. This included developing the necessary software and electronics infrastructure so that the arm can be controlled directly from our higher-level planner. This gives us a functional baseline, allowing us to expand for future research and development.

This report has described the process in which a lot of the low-level frameworks have been built, everything from setting up the electronics, developing the firmware, and creating the software stack. The current tests show the viability of the robot as a platform, indicating a fully functioning operation pipeline. Moreover, we have also performed a number of MRI testing with the sables and motors, laying the necessary groundwork for full MRI testing of the robot.
However, In the current iteration, we are still using relatively simple planning and control algorithms. Things like position-based PID and Moveit2's built in path planning libraries work to create a functional demonstration of the robot, but there are more effective alternatives which could be leveraged. Moreover, there are still a number of mechanical pain points that can be improved upon. The elbow joint experiences a significant amount of spring pre-load (and stress) during normal operation, making movement and sensing difficult. It would be beneficial to perform a partial re-design of the joint, especially with the possibility of using heavier end-effectors.

A number of these issues remain to be addressed as the project progresses, with the current timeline for the MRI robot project having it continue for another year. At this point, the low-level infrastructure have been completed, which allows for a significant amount of the robot's operation to be abstracted away. Thus, the future plans are to further fine-tune the robot's planning and control algorithms so that we have better defined and smoother actuation. With that complete, we will then perform the final in-MRI testing with all systems integrated.

Ultimately, this report has demonstrated the functionality of the MRI robot arm, developing the necessary infrastructure for the future continuation of this project.

# Bibliography


[1] D. J. Oathes, N. L. Balderston, K. P. Kording, *et al.*, "Combining transcranial magnetic stimulation with functional magnetic resonance imaging for probing and modulating neural circuits relevant to affective disorders," *WIREs Cognitive Science*, vol. 12, no. 4, e1553, 2021. DOI: https://doi.org/10.1002/wcs.1553. eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/wcs.1553. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wcs.1553.

[2] S. R. Kantelhardt, T. Fadini, M. Finke, *et al.*, "Robot-assisted image-guided transcranial magnetic stimulation for somatotopic mapping of the motor cortex: A clinical pilot study," en, *Acta Neurochir. (Wien)*, vol. 152, no. 2, pp. 333–343, Feb. 2010.

[3] J. G. Grab, E. Zewdie, H. L. Carlson, *et al.*, "Robotic TMS mapping of motor cortex in the developing brain," en, *J. Neurosci. Methods*, vol. 309, pp. 41–54, Nov. 2018.

[4] H. Shin, H. Jeong, W. Ryu, *et al.*, "Robotic transcranial magnetic stimulation in the treatment of depression: A pilot study," en, *Sci. Rep.*, vol. 13, no. 1, p. 14 074, Aug. 2023.

[5] S. Huang, C. Lou, Y. Zhou, *et al.*, "MRI-guided robot intervention—current state-of-the-art and new challenges," *Med-X*, vol. 1, no. 1, p. 4, 2023, ISSN: 2731-8710. DOI: 10.1007/s44258-023-00003-1. [Online]. Available: https://doi.org/10.1007/s44258-023-00003-1.

[6] Y. Koseki, T. Tanikawa, and K. Chinzei, "MRI-compatible micromanipulator; design and implementation and MRI-compatibility tests," in *2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2007, pp. 465–468. DOI: 10.1109/IEMBS.2007.4352324.

[7] H. Elhawary, A. Zivanovic, M. Rea, *et al.*, "The feasibility of MR-image guided prostate biopsy using piezoceramic motors inside or near to the magnet isocentre," en, vol. 9, no. Pt 1, pp. 519–526, 2006.

[8] J. Futterer, M. Schouten, T. Scheenen, and J. Barentsz, "Mr-compatible transrectal prostate biopsy robot: A feasibility study," *International Society for Magnetic Resonance in Medicine*, 2010.

[9] P. A., B. L., W. B., *et al.*, "A 3d-printed needle driver based on auxetic structure and inchworm kinematics," ser. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. Volume 5A: 42nd Mechanisms and Robotics Conference, Aug. 2018, V05AT07A057. DOI: 10.1115/DETC2018-85371. eprint: https://asmedigitalcollection.asme.org/IDETC-CIE/

proceedings-pdf/IDETC-CIE2018/51807/V05AT07A057/2476392/v05at07a057-detc2018-85371.pdf. [Online]. Available: https://doi.org/10.1115/DETC2018-85371.

[10] G. S. Fischer, I. Iordachita, C. Csoma, *et al.*, "MRI-compatible pneumatic robot for transperineal prostate needle placement," *IEEE/ASME Transactions on Mechatronics*, vol. 13, no. 3, pp. 295–305, 2008. DOI: 10.1109/TMECH.2008.924044.

[11] Z. Guo, Z. Dong, K.-H. Lee, *et al.*, "Compact design of a hydraulic driving robot for intraoperative MRI-guided bilateral stereotactic neurosurgery," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2515–2522, 2018. DOI: 10.1109/LRA.2018.2814637.

[12] N. Tsekos, J. Shudy, E. Yacoub, P. Tsekos, and I. Koutlas, "Development of a robotic device for MRI-guided interventions in the breast," in *Proceedings 2nd Annual IEEE International Symposium on Bioinformatics and Bioengineering (BIBE 2001)*, 2001, pp. 201–208. DOI: 10.1109/BIBE.2001.974430.

[13] R. Gassert, R. Moser, E. Burdet, and H. Bleuler, "MRI/fMRI-compatible robotic system with force feedback for interaction with human motion," *IEEE/ASME Transactions on Mechatronics*, vol. 11, no. 2, pp. 216–224, 2006. DOI: 10.1109/TMECH.2006.871897.

[14] F. Sergi, A. C. Erwin, and M. K. O'Malley, "Interaction control capabilities of an mr-compatible compliant actuator for wrist sensorimotor protocols during fMRI," *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 6, pp. 2678–2690, 2015. DOI: 10.1109/TMECH.2015.2389222.

[15] B. He, N. Zhao, D. Y. Guo, *et al.*, "Design and control of a compact series elastic actuator module for robots in MRI scanners," *arXiv preprint*, 2024. [Online]. Available: https://arxiv.org/pdf/2406.07670.

[16] A. D. Goyeneche, N. Zhao, C. Paxson, *et al.*, "Towards accurate and automated tms coil placement with a robotic system in MRI: A preliminary study," *International Society for Magnetic Resonance in Medicine*, 2025.

[17] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, eabm6074, 2022. DOI: 10.1126/scirobotics.abm6074. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.abm6074. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[18] M. Lohstroh, C. Menard, S. Bateni, and E. A. Lee, "Toward a lingua franca for deterministic concurrent systems," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 4, May 2021, ISSN: 1539-9087. DOI: 10.1145/3448128. [Online]. Available: https://doi.org/10.1145/3448128.

[19] E. A. Lee, R. Akella, S. Bateni, S. Lin, M. Lohstroh, and C. Menard, "Consistency vs. availability in distributed cyber-physical systems," *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 5s, Sep. 2023, ISSN: 1539-9087. DOI: 10.1145/3609119. [Online]. Available: https://doi.org/10.1145/3609119.