Implementing Adaptive Control in ROS 2 for a Laparoscopic Surgical Robotic Test Platform

Master Thesis

In partial fulfillment of the requirements for the degree

"Master of Science in Engineering"

Study program:

Mechatronics and Smart Technologies - Mechanical Engineering

Management Center Innsbruck

Supervisor:

FH-Prof. Yeongmi Kim, PhD

Author:

Liam Nolan

51843934

Declaration in Lieu of Oath

„I hereby declare, under oath, that this master thesis has been my independent work and has not been aided with any prohibited means. I declare, to the best of my knowledge and belief, that all passages taken from published and unpublished sources or documents have been reproduced whether as original, slightly changed or in thought, have been mentioned as such at the corresponding places of the thesis, by citation, where the extend of the original quotes is indicated.“

Place, Date	Signature

Acknowledgement

I want to thank....

Abstract

The advancement of laparoscopic surgical robotics has contributed significant progress to the field of minimally invasive surgery; however, their high cost limits widespread adoption in research and training efforts. This system utilizes low-cost, off-the-shelf hardware to produce an accessible development platform for research purposes. It leverages ROS 2's distributed framework to improve real-time system performance and enable future software architecture scalability. Advanced system identification methods were employed to characterize the system, and an adaptive control strategy was implemented to compensate for varying system dynamics throughout the range of motion. Experimental validation demonstrates that the ROS 2-based architecture, in combination with the advanced system identification and control methods, provides significant performance improvements over the previous implementation. The results confirm the feasibility of a cost-effective surgical robotic system with advanced, adaptable control strategies, contributing to the advancement and accessibility of the next generation of robotic-assisted surgical systems.

Keywords: Advanced control engineering, LQR controller

Contents

1 Introduction

1.1 Background and Context

Laparoscopic surgery offers a minimally invasive surgical option that significantly reduces postoperative pain and recovery time for patients. However, the complex dexterity demands of this surgical technique have created a need for robotic assistance. Systems such as Intuitive Surgical's da Vinci have been performing operations since 1999. While highly effective, these systems are prohibitively expensive and technically complex, and as a result, remain largely inaccessible to many researchers, surgeons, and clinicians. These barriers make it difficult to advance developments in the field of robotic-assisted surgery. In an effort to reduce these obstacles, researchers at MCI have previously developed a low-cost, desktop-operated surgical robotic system.

1.2 Problem Statement

While this system was effective in providing a low-cost alternative to existing solutions, its dynamic performance and software architecture left room for improvement. The software architecture was implemented using a C-based Arduino framework, which lacked many features necessary for supporting further technical advancements. Additionally, the system was controlled using a tuned PID controller which, while providing basic control over the system, left significant performance gains unrealized. The system was also highly nonlinear in nature, and as the control target deviated from the linearization point, its performance degraded considerably, resulting in inadequate control behavior.

1.3 Research Aim and Objectives

The goal of the research conducted in this thesis was twofold. First, to implement a ROS 2 framework for the existing system. ROS 2 is the current industry standard open-source distributed robotics middleware framework, making it well suited for the dynamic, precision-focused requirements of surgical robotic systems. Its modular nature allows for the development of multiple complex subsystems in parallel, and its framework can be easily adapted to accommodate future additions or improvements to the platform. Additionally, ROS 2 offers many essential tools for complex robotic systems, such as system visualization, collision detection, and frame transformation frameworks. Porting the existing system to this ROS 2-based framework would allow access to these benefits and facilitate further development of the system in the future.

Secondly, due to the lack of dynamic performance and control in the current system, this research aimed to develop a more robust characterization and control strategy to address its nonlinear nature. The improved characterization strategy involves comparing the results of step response experiments to closed-loop excitation data to better model the system. More advanced control strategies, such as Linear Quadratic Regulator (LQR) and Linear Quadratic Integral (LQI) controllers with feedforward control, could then be applied to this system model. Additionally, gain scheduling techniques would be utilized to adapt the controller to the system's highly nonlinear behavior.

1.4 Scope and Limitations

This thesis focuses on the implementation of an adaptive control framework within a ROS 2-based architecture for a laparoscopic surgical robotic test platform. The scope of the research includes the migration of the existing robotic system from a C-based Arduino framework to ROS 2, the development of advanced control strategies to address system nonlinearities, and the experimental validation of the proposed control methods using a laboratory-scale test platform.

Clinical application and in vivo testing are beyond the scope of this work; thus, the system is evaluated exclusively in controlled lab environments. The research prioritizes software architecture and control performance over hardware design improvements. Furthermore, the adaptive control methods are tested on predefined motion trajectories and system responses, which may not cover the full range of operational scenarios encountered in actual surgical procedures.

Limitations of the study include the use of a prototype platform that may not fully replicate the mechanical complexities of commercial surgical robots. The ROS 2 implementation is limited to core middleware functionalities, and some advanced features such as real-time communication optimizations and security enhancements are not fully explored. Additionally, due to resource constraints, long-term reliability and robustness tests were not conducted. These limitations highlight potential areas for future research to extend the applicability and performance of the system.

1.5 Thesis Structure

This thesis is organized into seven chapters. Chapter 1 introduces the research background, problem statement, and research objectives. Chapter 2 reviews the current state of the art in laparoscopic surgical robotics and control methods. Chapter 3 outlines the system design and methodology used to develop the ROS 2-based framework and adaptive control strategies. Chapter 4 details the implementation process, including software architecture and integration challenges. Chapter 5 presents the experimental results and performance evaluation of the developed system. Chapter 6 discusses the findings, limitations, and implications of the research. Finally, Chapter 7 concludes the thesis and proposes directions for future work.

2 Background and State of the Art

This chapter serves as an overview of the current state of the art regarding low-cost teleoperative surgical training systems. These systems are the current industry standard, and this section will explore their architecture, advantages, and disadvantages. Additionally, this section will delve into current control standards and strategies that these systems use and will discuss the Robotic Operating System 2 (ROS 2) and its applications.

2.1 The da Vinci Research Kit (dVRK)

The da Vinci Research Kit (dVRK) is a research platform developed through a collaboration between academic institutions, Johns Hopkins University and Worcester Polytechnic Institute, and Intuitive Surgical Inc. in 2012. It consists of first-generation da Vinci components that allow for a common development platform between institutions for the research of robotic-assisted surgery. The dVRK consists of the following components [1]:

- Two da Vinci Manual Tool Manipulators (MTMs)
- Two da Vinci Patient Side Manipulators (PSMs)
- A stereo viewer
- A foot pedal tray
- Manipulator Interface Boards (dMIBs)
- Basic accessory kit

At the time of writing in 2025 the DVRK program has been used by 40 research centers in 10 countries (citation needed)

2.2 Raven II

Section on background of Raven II

2.3 Robot Operating System (ROS 2)

Robot Operating System 2 (ROS 2) is a Linux-based, open-source robotics middleware suite consisting of software frameworks for robotic applications. ROS 2 is a direct evolution of ROS 1 and improves upon many aspects. While ROS 1 relied on a centralized ROS Master, ROS 2 uses the Data Distribution Service (DDS) for decentralized, peer-to-peer communication, eliminating the single point of failure present in the master node and increasing system reliability. ROS 2 also provides full real-time performance and supports most common microcontrollers through Micro-ROS, which is critical for many modern applications.

ROS 2 has proven itself as an industry standard, being utilized frequently in cutting-edge applications such as autonomous vehicles, humanoid robotics, and even surgical robotics (e.g., Intuitive Surgical is exploring its use for modular control—citation needed). Its suite of tools and utilities provides researchers with valuable development resources, including system visualization, collision detection, and frame transformation libraries, all of which are valuable assets to a Robotic Autonomous System (RAS).

2.4 Control Techniques in Robotic Surgery

Precise control, haptic feedback, complex system dynamics, and high safety standards drive the need for sophisticated control techniques in modern robotic surgery systems. Industry-leading platforms such as Intuitive Surgical's da Vinci system utilize Model Predictive Control (MPC) (Kazanzides et al., 2011), enabling comprehensive modeling and prediction of system behavior. These systems also employ hierarchical control architectures (combining high-level planning with low-level execution) along with adaptive learning-based controllers to compensate for tissue deformation and operate effectively in dynamic environments (IEEE Reference).

Surgeon feedback is equally critical for successful surgical outcomes. Advanced haptic feedback and force control algorithms are implemented in master-tool manipulators (MTMs) to provide surgeons with precise tactile information. Furthermore, real-time imaging and navigation systems deliver valuable intraoperative feedback to enhance surgical precision.

The integration of artificial intelligence has revolutionized surgical robotics in recent years. Deep learning approaches now enable advanced perception and decision-making capabilities within these systems. For instance, real-time tissue identification (arXiv:2305.07841) allows for dynamic force modeling, enabling systems to adapt intelligently to varying tissue properties during procedures.

2.5 System Identification Techniques in Robotic Surgery

Implementing advanced control strategies requires a dynamic understanding of system behavior, which can only be achieved through robust system identification. The highly nonlinear nature of many robotic systems demands excitation methods that remain effective under nonlinear conditions. Multisine and chirp signals form the basis for many modern system identification techniques (Mechatronics, 2022: https://www.sciencedirect.com/science/article/abs/pii/S0957415822000381), and are increasingly combined with reinforcement learning approaches (IEEE CDC, 2023: https://ieeexplore.ieee.org/document/10383476).

In surgical robotics, where safety is paramount, a multi-stage identification process is critical. For example, Intuitive Surgical performs low-level actuator characterization using step-response tests to validate settling time, overshoot, and stiffness in every motor and joint (Patent US20230309921A1: https://patents.google.com/patent/US20230309921A1). These physics-based models are then enhanced with AI/ML layers to handle complex surgical interactions (Patent US11432888B2: https://patents.google.com/patent/US11432888B2). This hybrid approach delivers uncompromising system dynamics and performance.

2.6 Summary and Research Motivation

Modern robotic-assisted surgical systems provide a powerful framework to assist doctors and surgeons, enabling them to improve patient quality of care while reducing postoperative recovery time. While these systems are incredibly capable, their high cost and barriers to entry have limited accessibility for researchers. This limitation has led to the development of several lower-cost alternatives, such as the Da Vinci Research Kit, Raven II, and the Desktop Surgical System.

These systems inherit many features found in commercial solutions, including a ROS-based architecture and advanced control and identification techniques. However, the gap between their implementation and the current state of the art remains apparent. This research aims to reduce this technical gap by:

- Upgrading the software architecture to ROS 2,
- Implementing more advanced system identification techniques, and
- Incorporating adaptive control to improve system performance.

3 Desktop Teleoperated Surgical Training System

In an effort to further advance the field of robotic-assisted surgery, researchers at MCI have developed a Desktop Teleoperated Surgical Training System. The system's design was heavily inspired by the da Vinci Research Kit (DVRK) and Raven II, with a focus on maximizing performance while minimizing cost. This system serves as the foundation for the research outlined in this paper, so an understanding of its design and architecture is essential.

3.1 Overall System Architecture

The system consists of two main components:

- The Master Tool Manipulator (MTM)
- The Patient Side Manipulator (PSM)

3.1.1 Master Tool Manipulator (MTM)

The MTM is a serial-link manipulator that measures the operator's motion through a 7-DOF (degree-of-freedom) linkage, actuated by the user. It includes:

- 3 DOFs for tracking overall system position (J0–J2)
- 4 DOFs for tracking gimbal position and state (G0–G3)

This allows the operator to communicate desired motions to the PSM. The J0–J2 DOFs dictate the endpoint position, while G0–G3 control endpoint orientation and state.

The MTM should allow for accurate interpretation of the operator's intended motion while also minimizing the force perceived by the operator. This perceived force requirement drove the design toward a gravity compensation system (GCS).

3.1.2 Patient Side Manipulator (PSM)

The PSM has 7 primary DOFs, divided into:

- **Overall System Position:**
  - Roll
  - Pitch
  - Insertion
- **Surgical Instrument Control:**
  - Instrument Roll
  - Instrument Pitch
  - Instrument Tilt
  - Instrument Open/Close

The system is actuated using:

- Maxon brushless motors (for overall positioning)
- Servo motors (for instrument control)

Position feedback is provided by three 12-bit absolute optical encoders, and force reduction is achieved through:

- Capstan drives (for roll and pitch axes)
- Rack-and-pinion (for insertion)

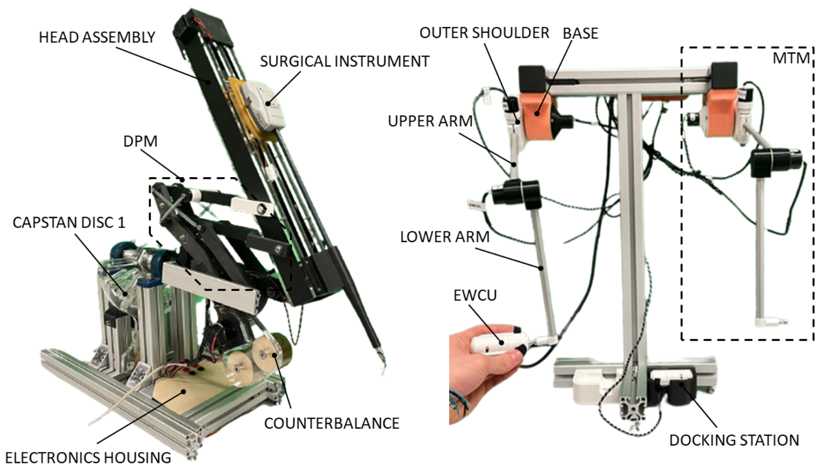

Figure 3.1: Placeholder image of system

3.2 MTM Mechanical Design

The Master Tool Manipulator (MTM) can be separated into two subsystems:

- Arm (J0-J2)

- Gimbal (G0-G3)

The arm subsystem refers to the overall architecture which is responsible for measuring the operator's position with respect to (w.r.t.) the global reference frame. Three revolute joints with internal optical encoders are coupled via rigid aluminum extrusions, allowing for accurate estimation of the overall system positioning in 3D space, as shown in Figure 3.2.

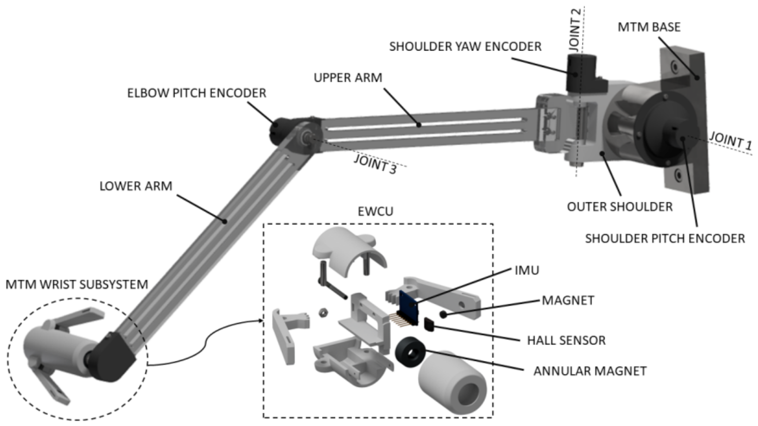

Figure 3.2: Detailed depiction of the MTM's arm and wrist subsystems, with their individual components labeled.

3.2.1 Joints

The arm of the manipulator consists of three revolute joints connecting the four links of the MTM arm. Table 3.1 summarizes these joints and provides a description of their function.

Table 3.1: Summary and description of revolute (R) joints of the MTM's arm [2, 3].

MTM Joint	Joint Type	Joint Name	Description
1	R	Shoulder pitch	This joint moves the entire MTM and carries out pitch rotation similar to joint 3.
2	R	Shoulder yaw	This joint is responsible for adduction and abduction movement of the operator.
3	R	Elbow pitch	This joint is the second pitch joint of the MTM. Together with joint 1, it is responsible for up/down and forward/backward motion.

The design of these joints was driven by the requirement to minimize friction. Increased frictional forces in any of these joints would result in an increase in perceived force by the operator. As visible in Figure 3.3, roller bearings were utilized to minimize these frictional forces, and system mass was also reduced to lower both gravitational and inertial forces felt by the operator.

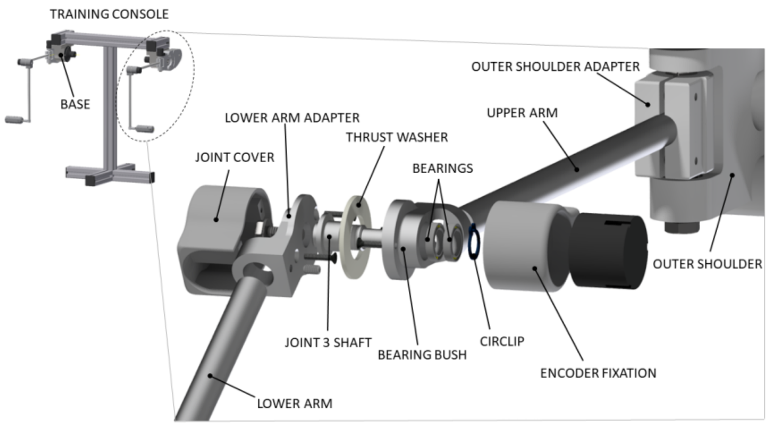

Figure 3.3: Exploded view of the MTM components (excluding EWCU fixation) showing the updated training console design. Key improvements include: (1) circlip for axial safeguarding, (2) PTFE thrust washer to eliminate bearing clearance effects while maintaining low friction, and (3) reduced wear characteristics.

3.3 MTM Electrical Architecture

3.3.1 System Overview

The MTM (Master Tool Manipulator) electrical architecture was originally based around an Arduino Mega 2560 microcontroller that served as both the primary controller and system I/O interface. This configuration was later upgraded to a Teensy 4.1 microcontroller due to its superior performance and compatibility with ROS2 and micro-ROS frameworks.

The system incorporates multiple sensor subsystems to monitor and control the robotic arm's movements. Each side of the arm is equipped with sensors for:

- Shoulder pitch (J0)
- Shoulder yaw (J1)
- Elbow pitch (J2)
- Wrist orientation (G0-G2)
- Grasper state (G3)

3.3.2 Joint Subsystems (J0-J2)

Joints J0-J2 are equipped with AEAT-6012-A06 12-bit absolute magnetic encoders (resolution: 4096 counts/revolution) directly coupled to each joint axis. These encoders provide a positional accuracy of $\pm 0.088°$ $(360°/4096)$ per joint, enabling precise positional feedback for the control system.

- **Shoulder Pitch (J0)**: Magnetic encoder provides precise feedback on shoulder pitch position
- **Shoulder Yaw (J1)**: Magnetic encoder measures shoulder yaw angle
- **Elbow Pitch (J2)**: Magnetic encoder monitors elbow pitch movement

The encoder signals are connected directly to the Teensy 4.1's digital I/O pins through a noise-resistant shielded cable assembly.

3.3.3 Wrist Subsystem (G0-G3)

The wrist subsystem underwent significant design evolution:

- **Initial Design**: Utilized an MPU-6050 IMU (3-DOF) for wrist orientation sensing and a Hall effect sensor for grasper state detection
- **Issues Encountered**: Noticeable drift occurred in the IMU during prolonged operation
- **Final Implementation**: Replaced with analog potentiometers for wrist position detection (G0-G2) while retaining the Hall effect sensor for grasper state detection (G3)

The MPU-6050 IMU provided the following measurement capabilities:

- 3-axis accelerometer ($\pm 2/4/8/16$ g configurable)
- 3-axis gyroscope ($\pm 250/500/1000/2000°$/s configurable)

3.3.4 Connectivity

All wrist sensors (G0-G3) connect to the Teensy 4.1's analog I/O pins through a 7-strand flat flexible cable (FFC) with the following specifications:

- Pitch: 0.5mm
- Current rating: 0.3A per conductor
- Voltage rating: 50V
- Temperature range: $-40°$C to $+105°$C

3.3.5 Grip Subsystem

The grip detection system utilizes a Hall effect sensor to measure the position between the index finger and thumb components. Key characteristics include:

- Sensing range: ± 50mm
- Resolution: 0.1mm
- Output: Analog voltage (0-3.3V)

3.4 PSM Mechanical Design

3.4.1 Overview

The Patient-Side Manipulator (PSM) mechanical architecture is a 7-degree-of-freedom (DOF) robotic system designed to replicate surgical instrument movements with high precision. The design is fundamentally inspired by the da Vinci Surgical System's remote center of motion (RCM) mechanism, which maintains a fixed pivot point at the surgical incision site - a critical feature for minimally invasive surgery. The PSM can be divided into five primary subsystems:

- Superstructure and base
- Roll (yaw) axis
- Pitch axis
- Insertion axis
- Tool cart and end effector

3.4.2 Superstructure

The superstructure serves as the foundational framework, constructed from 80/20 aluminum extrusion for both rigidity and modularity. This design provides:

- Structural stability for the entire system
- Mounting points for all electromechanical components
- Vibration damping through its triangulated geometry
- Accessibility for maintenance and upgrades

The aluminum extrusion profile (40mm × 40mm, series 8) was selected for its excellent strength-to-weight ratio and compatibility with standard motion components.

3.4.3 Roll (Yaw) Axis

The roll axis implements a revolute joint providing $\pm 90^\circ$ of rotation about the vertical axis, corresponding to the da Vinci system's first DOF.

- **Drive System**: Maxon EC-4pole 30 brushless DC motor (50W) coupled to a 20:1 Capstan drive
- **Transmission**: Zero-backlash Capstan drive (0.1 arc-minute repeatability)
- **Bearing System**: Two NSK LH-series linear bearing blocks supporting a 16mm ground aluminum shaft

- **Position Sensing**: Broadcom HEDS-5540 optical encoder (500 CPR)
- **Safety**: Omron D2F-L-A1 limit switches at $\pm 90^\circ$ travel limits

The roll axis terminates in an aluminum fork structure that supports both the pitch and insertion axes, maintaining the kinematic chain.

3.4.4 Pitch Axis

The pitch axis replicates the da Vinci's characteristic parallelogram mechanism, which creates a virtual RCM approximately 50mm distal to the joint - matching typical trocar insertion depths.

- **Mechanism**: Dual-parallelogram linkage with 3D-printed ABS components
- **Drive System**: Maxon EC-4pole 30 motor with identical 20:1 Capstan drive
- **Range of Motion**: $\pm 45^\circ$ from neutral position
- **Advantages**:
  - Maintains constant instrument orientation during arm motion
  - Creates natural RCM without complex constrained mechanisms
  - Reduces shear forces at the incision point

The 3D-printed design (Prusa MK3S+, 0.15mm layer height) was chosen for rapid prototyping and weight reduction, with critical joints reinforced with stainless steel pins.

3.4.5 Insertion Axis

The prismatic insertion axis provides 200mm of linear travel, corresponding to the da Vinci's third DOF.

- **Drive Mechanism**: HV7346MG servo (modified for continuous rotation) with rack-and-pinion transmission
- **Guide System**: IGUS drylin linear rails (model RJ4JP-01-08)
- **Position Sensing**: Integrated optical encoder with 0.1mm resolution
- **Safety**: Dual Omron limit switches at travel extremes

The rack-and-pinion system (module 0.5, 20° pressure angle) provides 96.5% efficiency while maintaining the compact form factor required for surgical applications.

3.4.6 Tool Cart and End Effector

The tool cart integrates four degrees of freedom in a package matching Intuitive Surgical's instrument specifications:

- **Actuation**: Four MG966R servos with spring-loaded coupling mechanisms
- **Functions**:
  - Instrument roll ($\pm 90°$)
  - Wrist pitch ($\pm 85°$)
  - Wrist yaw ($\pm 85°$)
  - Grasper actuation (0-100%)
- **Instrument Interface**: Compatible with Intuitive Surgical EndoWrist® tools
- **Coupling Mechanism**: Fail-safe magnetic coupling with 5N retention force

3.4.7 Remote Center of Motion Implementation

The PSM's mechanical architecture replicates the da Vinci system's key innovation - the passive RCM mechanism. This is achieved through:

- **Parallelogram Linkage**: Mathematically constrains motion to a fixed point
- **Kinematic Design**: Aligns axes to intersect at the RCM point
- **Clinical Benefits**:
  - Minimizes tissue trauma at incision site
  - Provides intuitive instrument control
  - Reduces fulcrum effect during manipulation

3.5 PSM Electrical Architecture

3.5.1 System Overview

The Patient-Side Manipulator (PSM) electrical architecture is designed to meet stringent surgical robotic requirements, including:

- Sub-millimeter positional accuracy (± 0.2mm at instrument tip)
- High dynamic response (bandwidth > 10Hz for all axes)
- Fail-safe operation with redundant position verification

- Low-latency control loops ($<$1ms cycle time)

The system implements a distributed control architecture centered around a Teensy 4.1 microcontroller (600MHz ARM Cortex-M7), chosen for its:

- Real-time performance capabilities
- Extensive I/O options (28 PWM channels, 16 analog inputs)
- Native USB 2.0 support for ROS2 integration
- FPU support for advanced control algorithms

3.5.2 Position Sensing System

Primary Joint Sensing

- **Roll/Yaw Axis**: Broadcom HEDS-5540 optical encoder (500 CPR) with quadrature decoding
- **Pitch Axis**: Broadcom HEDS-5540 optical encoder (500 CPR)
- **Insertion Axis**: AMS AS5047P magnetic encoder (14-bit resolution, 0.022° accuracy)

Secondary Verification

- Omron D2F-L-A1 mechanical limit switches at all travel extremes
- Current sensing via INA219 shunt monitors (0.5% accuracy)
- Thermal monitoring on all motor drivers

3.5.3 Motor Drive System

Main Axes Actuation

Table 3.2: Main Axes Drive Specifications

Parameter	Roll	Pitch	Insertion
Motor	Maxon EC-4pole 30	Maxon EC-4pole 30	HV7346MG
Voltage	24V	24V	12V
Continuous Current	2.5A	2.5A	3.2A
Peak Torque	50mNm	50mNm	15kg-cm
Driver	MD13S Cytron	MD13S Cytron	MD13S Cytron
Control Mode	PID Velocity	PID Position	PID Position
Bandwidth	500Hz	500Hz	300Hz

Servo Control Implementation

The wrist subsystem utilizes four MG966R servos modified for continuous rotation and external position control:

- **Control Method**: PWM (1-2ms pulse width) with 12-bit resolution
- **Position Feedback**: Internal potentiometer (10kΩ, 300° range)
- **External Verification**: AS5600 magnetic encoders on output shafts
- **Spring Coupling**: 0.5N/mm preloaded springs ensure zero backlash

3.5.4 Safety and Calibration

The system implements a comprehensive safety architecture:

- Dual-channel limit switch verification (mechanical + optical)
- Watchdog timer (100ms timeout) on all motor drivers
- Current limiting at both driver and software levels
- Automatic homing sequence on startup:
  1. Seek limit switch at reduced velocity (10% max) *Back off 5mm from limit*
  2. Establish encoder zero position

3.5.5 Control Architecture

The Teensy 4.1 implements the following control scheme:

- **Sampling Rate**: 1kHz for all critical loops
- **Algorithm**: Adaptive PID with velocity feedforward
- **Communication**:
  - ROS2 interface (micro-ROS) @ 100Hz
  - CAN bus for motor drivers (1Mbps)
  - I^{2}C for sensors (400kHz fast mode)
- **State Estimation**: Kalman filter combining encoder and IMU data

3.5.6 Power Distribution

- **Main Bus**: 24V DC (Mean Well LRS-350-24)
- **Logic Power**: 5V/3A buck converter (TI TPS5430)
- **Protection**:
  - Polyfuses on all outputs
  - TVS diodes for surge protection
  - Star grounding for noise reduction

3.6 System Software Architecture

As noted earlier in this paper, the initial software framework was implemented in an Arduino C-based environment. While the system has since been migrated to a ROS 2 architecture, the original implementation remains relevant for context and reference.

The software was divided into two primary components: the **Master Tool Manipulator (MTM)** software and the **Patient Side Manipulator (PSM)** software. These subsystems worked in tandem to enable real-time teleoperation, with the MTM capturing user inputs and the PSM replicating movements precisely.

3.6.1 MTM Software

The MTM software was responsible for:

1. **Sensor Data Acquisition**:
   - Reading six absolute magnetic encoders, two IMUs (Inertial Measurement Units), and two Hall effect sensors (eight sensor values per MTM arm)
   - Distinguishing between left and right-arm data for independent kinematic processing
2. **Forward Kinematics**:
   - Calculating the system's position and orientation from raw sensor data
   - Smoothing noisy signals using low-pass filters (`LPFilter.ino`, `LPFilter_Encoder.ino`)
3. **Data Processing and Safety**:
   - Detecting abrupt movements via spike detection logic (`spikeDetection.ino`)
   - Monitoring sensor overflow conditions (`OverFlowDetection.ino`)

4. **Communication**:
    - Relaying processed kinematic data to the PSM via serial communication
    - Packaging data with start/end markers for robust parsing (`recvWithStartEndMarkers.ino`)

Key files included:

- `MT_MTM.ino`: Central coordination, subsystem integration, and communication with the Arduino Mega
- `KinematicCalc_L/R.ino`: Arm-specific forward kinematics for joint angle derivation
- `readIMU.ino` & `readEncoder.ino`: IMU quaternion and encoder position data handling

3.6.2 PSM Software

The PSM software replicated MTM movements using:

1. **Data Reception and Parsing**:
    - Extracting target positions from serial messages (`parseData.ino`)
2. **Closed-Loop Control**:
    - Executing PID control for motor actuation (`PIDupdate.ino`)
    - Implementing ramp-up routines for smooth motion initiation (`RampUp_Homing.ino`)
3. **Safety and Calibration**:
    - Collision detection via limit switches (`CheckCol.ino`)
    - Reference drives for incremental encoders (joints 1–3) to establish absolute positioning post-startup
    - Tool manipulation routines (`Catch_Tool.ino`) and PWM frequency tuning (`setPwmFrequency.ino`)

3.7 System Kinematics and Coordinate Frames

3.8 System PID Controller Design

4 Implementation on Current Hardware

4.1 Software Architecture

This section will serve as an overview of the current system software architecture and will provide an in-depth analysis of how the current ROS2 Node structure works.

4.1.1 ROS 2 System Architecture

The current system's ROS 2 architecture is composed of four primary nodes, each responsible for specific functionalities:

- **MTM Node (Micro-ROS):** This node is responsible for streaming sensor data from the Master Tool Manipulator (MTM) and performing forward kinematics calculations to determine the target pose.
- **System Model Node:** This node maintains a Simulation Description Format (SDF) model of the MTM. It facilitates real-time visualization of the system's configuration and enables the calculation of various joint frames.
- **Joint Publisher Node:** Acting as an intermediary, this node utilizes the SDF model to offer an alternative method for calculating the target position based on TF (Transform Frame) transformations.
- **PSM Node (Micro-ROS):** This node handles sensor data acquisition, implements control logic, and drives the Patient Side Manipulator (PSM) system.

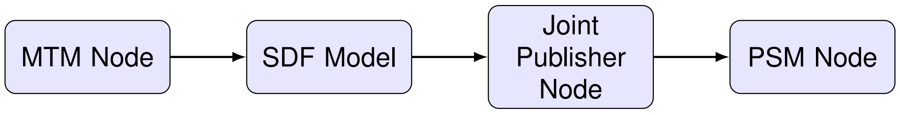

Figure 4.1: Initial ROS 2 System Architecture for Target Pose Calculation.

Initially, the research aimed for the MTM Node to relay its joint angles to the SDF Model of the MTM. Subsequently, the Joint Publisher Node would calculate the desired end-effector position via a TF frame transformation from the base frame to the gimbal origin. This approach was intended to provide an alternative method for extracting the target position, bypassing the need for explicit forward kinematics in the MTM node. This architecture is illustrated in Figure 4.1. While this method proved effective in accurately calculating the target pose, real-time performance trials revealed significant limitations.

Specifically, although the target position extracted from the model demonstrated excellent real-time performance, a communication protocol breakdown was observed when the Joint Publisher Node interfaced with the Micro-ROS PSM Node. This breakdown was likely caused by memory limitations inherent to the Teensy 4.1 microcontroller, leading to high latency and noticeable data stuttering.

—

Consequently, a simplified architecture was explored to overcome these real-time performance issues. This revised approach involves performing forward kinematics calculations of the target position directly within the MTM Node and streaming this data topic directly to the PSM Node. This simplification of the technical stack significantly improved real-time performance and reduced system complexity. This optimized architecture is presented in Figure 4.2.

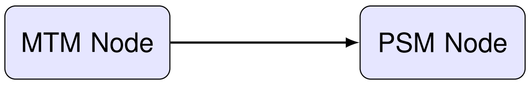

Figure 4.2: Optimized ROS 2 System Architecture for Direct Target Pose Streaming.

4.1.2 MTM Modeling and Visualization

For the modeling of the MTM, the Simulation Description Format (SDF) was chosen due to its compatibility with ROS 2 and RViz. SDF also supports complex kinematic structures, including closed-chain kinematics, which is crucial for accurately modeling the more intricate four-bar style linkages found in the Patient Side Manipulator (PSM).

Generating the SDF file proved to be a fairly involved process. First, the CAD model of the MTM system was imported into SolidWorks. To reduce computational complexity, the model underwent significant simplification. Parts were condensed into their respective links, identified as follows:

- Base
- J1 (Joint 1)
- J2 (Joint 2)
- J3 (Joint 3)
- G3 (Gimbal Link 3)
- G2 (Gimbal Link 2)
- G1 (Gimbal Link 1)
- G0 (Gimbal Link 0)

Each of these components was created as a singular solid part and then reassembled within SolidWorks into a complete assembly.

Following this, rotational axes and origins were assigned for each link. A SolidWorks plugin was then utilized to export the model into a Universal Robot Description Format (URDF) file. This URDF file was subsequently converted to SDF using a dedicated software tool, *ros2_sdf_to_urdf* (further citation needed). Upon the successful generation of the SDF model, a Python launch file was developed to facilitate quick and easy visualization of this model within RViz.

The final result of this effort can be seen below in Figure 4.3.

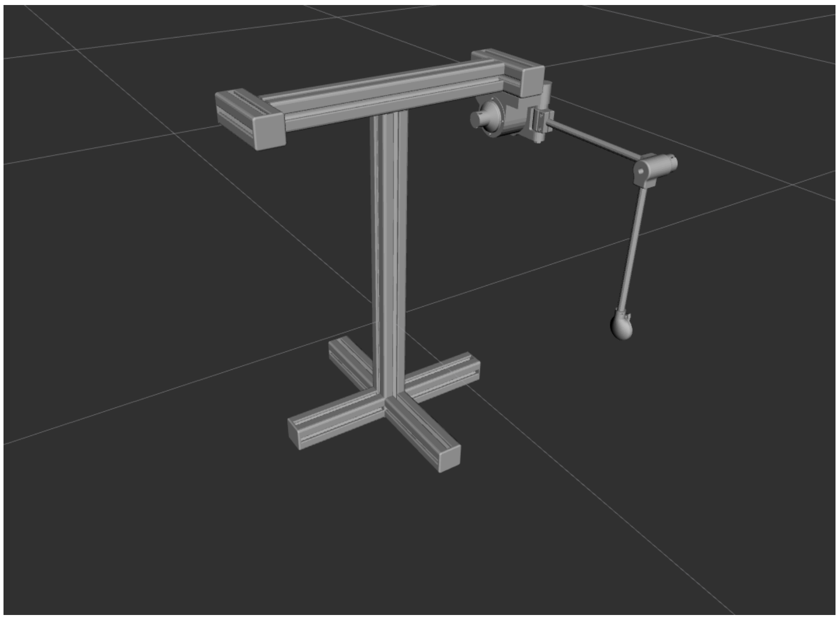
Figure 4.3: SDF model of MTM

4.1.3 MTM Data Streaming and filtering

The MTM node's primary purpose is to process incoming sensor data and broadcast it into the `ROS2` workspace in a usable format. The following topics are published into the `ROS2` workspace from the MTM:

—

The MTM node's primary purpose is to process incoming sensor data and broadcast it into the `ROS2` workspace in a usable format. The following topics are published into the `ROS2` workspace from the MTM:

—

Table 4.1: MTM ROS2 Topics

Topic Name	Topic Type	Description
mtm_joint_states	sensor_msgs/msg/JointState	Current angular positions for all robotic arm joints.
mtm_raw_values	std_msgs/msg/Float32MultiArray	Unfiltered raw sensor data for debugging.
12target_pose	sensor_msgs/msg/JointState	Calculated Cartesian (x, y, z) position of the end-effector in meters.

4.1.4 MTM to PSM Communication Protocol

As previously established in Section 4.1, there are two separate paths for data communication between the MTM and the PSM: the direct communication path and the path utilizing the SDF model as an intermediary.

A critical aspect of both communication paths is ensuring **Quality of Service (QoS)** protocol matching. QoS allows for the definition of how topics are communicated between nodes. Ensuring that communicating nodes utilize appropriate and compatible QoS protocols is essential for software reliability and robustness. For this research, the QoS reliability and queue depth were modified to ensure proper message delivery.

Initially, an issue arose with the communication protocol between the SDF model and the PSM node. Data streamed from the SDF model was published to the `/model_pose` topic. When this data was plotted, it exhibited high quality with zero latency or lag. However, when the PSM topic `/psm_joint_telemetry` subscribed to `/model_pose`, a significant loss of data occurred, resulting in lag and stuttering. This led to unacceptable system performance. After extensive diagnostics, the data performance was improved by drastically increasing the publishing rate from 100 Hz to 1000 Hz, increasing the queue depth from 10 to 20, and applying the following QoS profile:

- **Reliability:** `RELIABLE`
- **History (Depth):** 20
- **Durability:** `VOLATILE`
- **Lifespan:** Infinite
- **Deadline:** Infinite
- **Liveliness:** `AUTOMATIC`
- **Liveliness lease duration:** Infinite

Increasing the publishing rate to such a drastic level required careful memory consideration to avoid overwhelming the limited memory of the PSM's Teensy 4.1. As a result, the `model_pose` message type was changed from a `JointTelemetry` message type to a `JointState` message type to reduce memory strain.

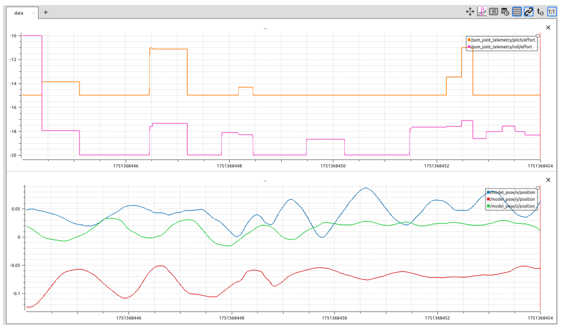

Figure 4.4: Enter Caption

4.1.5 PSM Software

The PSM software node's primary responsibility is the control of the physical PSM system. While this task may appear simple on the surface, its implementation was quite complex, requiring several different software components to work together cohesively.

The PSM node had two primary modes:

- **Target Mode:** In this mode, the PSM node received target joint angles from the MTM node and the Joint Publisher Node. The PSM would then drive the system to these target angles using a control algorithm. This mode was primarily used for the normal operation of the PSM.
- **Verification Mode:** In this mode, the PSM followed a predetermined trajectory to verify the system's performance. This mode was primarily used for testing and validation purposes.

In both modes, the primary codebase remained the same, consisting of a `main.cpp` file and multiple helper files. The primary difference between the modes was the subscription to the target pose topic. In Target Mode, the PSM node subscribed to the `/mtm_target_pose` topic, while in Verification Mode, it generated its own target pose trajectory and was completely self-contained.

In Target Mode, the PSM node subscribed to two different topics. For overall positioning of the system, it subscribed to the `/target_pose` topic generated by the Joint Publisher Node. This topic provided the target position in an `x, y, z` format. The PSM node then used this target position to calculate the necessary joint angles to achieve this position. Additionally, to determine the desired tool angles, it subscribed to the `/mtm_joint_states` topic. This topic provided the current gimbal angles of the MTM system, which were used to calculate the necessary joint angles for the PSM system.

In an effort to modularize the codebase and improve readability the PSM software was broken up into several different files, each responsible for a specific functionality. The following is a list of the primary files in the PSM software codebase:

- **config.h**: The file acts as a central repository for all configuration parameters used throughout the PSM software. It contains definitions for various constants, such as the number of axes, pin definitions, motor IDs, and other parameters that can be easily modified to adjust the system's behavior.

- **LQR.cpp**: This file implements the Linear-Quadratic Regulator (LQR) and Linear-Quadratic-Integral (LQI) control algorithms. These controllers are used to calculate the necessary control inputs for the roll and pitch axes of the robotic arm, ensuring stable and precise movements.
- **PID.cpp**: This file contains the implementation of a standard PID (Proportional-Integral-Derivative) controller. It provides an alternative control strategy that can be used for the various axes of the robot, calculating control values based on the error between the target and actual positions.
- **axis_conversion.cpp**: This file provides functions to convert between encoder counts and physical units such as angles (degrees) and linear positions (mm). This is essential for interpreting sensor data and commanding the motors accurately.
- **encoder_lp.cpp**: This file implements a low-pass filter for the encoder readings. This is used to smooth out noisy encoder signals, providing a cleaner and more stable measurement of the robot's position.
- **encoder_reader.cpp**: This file is responsible for reading the raw data from the motor encoders. It defines the encoder objects and provides a function to read the encoder counts for each axis.
- **gain_schedule.cpp**: This file implements a gain scheduling algorithm for the LQR-based controllers. It contains precomputed gain tables and uses interpolation to select the appropriate controller gains based on the current roll and pitch angles of the robot.
- **helper_functions.cpp**: This file contains various utility functions used throughout the project. These include functions for publishing debug messages, mapping gimbal angles to servo angles, and handling critical errors.
- **homing.cpp**: This file implements the homing sequence for the motors. This routine is responsible for moving the robot to a known starting position, which is crucial for accurate and repeatable operation.
- **main.cpp**: This is the main entry point of the application. It initializes the ROS 2 node, sets up subscribers and publishers, and contains the main control loop that reads sensor data, calculates control inputs, and commands the motors.
- **psm_joint_angles.cpp**: This file contains the inverse kinematics calculations for the patient-side manipulator (PSM). It computes the required joint angles for the roll, pitch, and insertion axes based on a target position in Cartesian space.
- **ramp.cpp**: This file provides a function to generate a smooth motion profile, or ramp, between a current and target angle. This is used to prevent jerky movements and ensure the robot moves in a controlled manner.

4.2 System Identification

For this research, proper controller design was paramount. However, a high-performance controller could not be designed without a fundamental understanding of the system's dynamics. To achieve this, a robust system identification scheme was developed.

The two primary joints of interest were the roll and pitch joints. These joints exhibited complex system dynamics and a high degree of nonlinearity. An exact system identification approach was tailored to each joint's unique dynamics.

4.2.1 Roll Joint Open loop Step Response

The roll joint can be mechanically described as a pendulum. Perturbations applied to the system move the joint away from its 0° position, but when the perturbation ceases, gravitational forces return the joint to its 0° position. The system was also determined to have a fairly consistent and predictable transfer function. Specifically, when the brushless motor was driven at a specific voltage, the resulting positional change of the system was consistent across trials. As a result of these factors, a step response characterization was deemed an excellent method for system identification.

A step response characterization is a common system identification method in control engineering. In this technique, the system's input is abruptly changed from one value to another, and the resulting system response is measured and used to characterize the system's dynamics.

For the roll axis, a very lightly tuned PID controller was initially employed to drive the system to the intended point of characterization. A predetermined delay allowed the system to settle into a stable state. Upon completion of this delay, the current motor voltage value was stored, then scaled by a factor of 1.1 before being sent to the motor. Both the motor voltage input over time and the positional output of the roll joint over time were recorded. This system response data was saved as a CSV file and subsequently loaded using a custom MATLAB script. This script parsed the data and then fitted a second-order transfer function to the system response using MATLAB's `tfest` command. Finally, state-space matrices could then be extracted from this transfer function using MATLAB's `ss` command.

4.2.2 Closed loop excitation through PRBS

Initial characterization of the pitch subsystem used a similar approach to the roll subsystem. A step response was applied, and the resulting response was used to characterize the system. However, several issues were found with the step response characterization of the pitch subsystem.

Due to the mechanical nature of the joint, the pitch subsystem's response was highly nonlinear. A slight change in voltage applied to the motor resulted in large changes in the pitch system's position, and this behavior was not consistent between trials. The same change in motor voltage would not result in the same system response. Additionally, when a small increase in voltage was applied to a stable system position, the pitch subsystem would drive to the end of its range of motion and contact the hard stop. This overshooting was a result of the pitch subsystem's extremely sensitive system response as its angle increased. This sensitivity was a direct consequence of the mechanical design of the pitch subsystem, where the force applied by the gravity compensation system drastically reduced as the angle of the pitch subsystem increased.

As a result of these complexities, a different approach had to be implemented. Due to the difficulties in maintaining the pitch subsystem within its intended range of motion, it was deduced that system identification needed to be conducted under closed-loop system control.

For this system identification technique, a light PD controller was applied to the system. The system was then commanded to its intended characterization point. Upon a predetermined settling time, the control input was then excited with a Pseudo-Random Binary Sequence (PRBS).

PRBS is a control signal excitation technique in which the control input is randomly fluctuated by a set amplitude at pseudo-random, predetermined intervals. By exciting the control input, the resulting behavior and system dynamics could be observed while the aforementioned PD controller attempted to bring the system back to its setpoint. Key considerations in closed-loop PRBS excitation are ensuring that the amplitude of the excitation is sufficient to bring the system off its setpoint and overcome noise, while also ensuring that the amplitude of the excitation does not overpower the controller and drive the system off its setpoint. Additionally, the intervals between the fluctuations of the excitation need to be chosen such that the system's dynamics are excited across a variety of frequencies.

To ensure that the system undergoes excitation at all critical frequencies multiple times, the PRBS characterization was run for an extended period. Trials typically lasted 30-660 seconds. During these trials, the pitch subsystem's input voltage and resulting position were continuously recorded. This output data was saved as a CSV file, and an additional MATLAB script was used to perform a mathematical characterization of the system.

Similarly to the roll subsystem, the recorded output of the pitch subsystem was loaded into a MATLAB script, and a variety of system identification techniques were applied to the data.

From these results, it was determined that a 4th-order transfer function estimation was the best method for characterizing the system.

After a 4th-order transfer function was fitted to the system using the MATLAB `tfest` command, the state-space matrices were extracted using the `ss` command. The estimated model's accuracy was then determined by comparing the actual system behavior based on the input to the predicted system behavior based on the system input using the MATLAB `compare` command.

From this comparison, it was determined that the model accuracy was quite low when compared to the real system response based on the input. However, when a controller was designed around this model estimation, the actual system performance was quite good. So, while this poor system estimation may point towards some underlying issues in the system identification technique, it ultimately resulted in a high-performing controller, which was the ultimate goal of this research.

4.2.3 Multi-point System Identification

Due to the kinematics of the system, the force required of the motors at various joint positions will vary. As the roll angle diverges farther from 0 in both the negative and positive direction, there is an increase in gravitational forces seen by the actuators, and as a result, the voltage applied to the motor would need to be increased accordingly. This trend is theoretically illustrated in Figure 4.5.

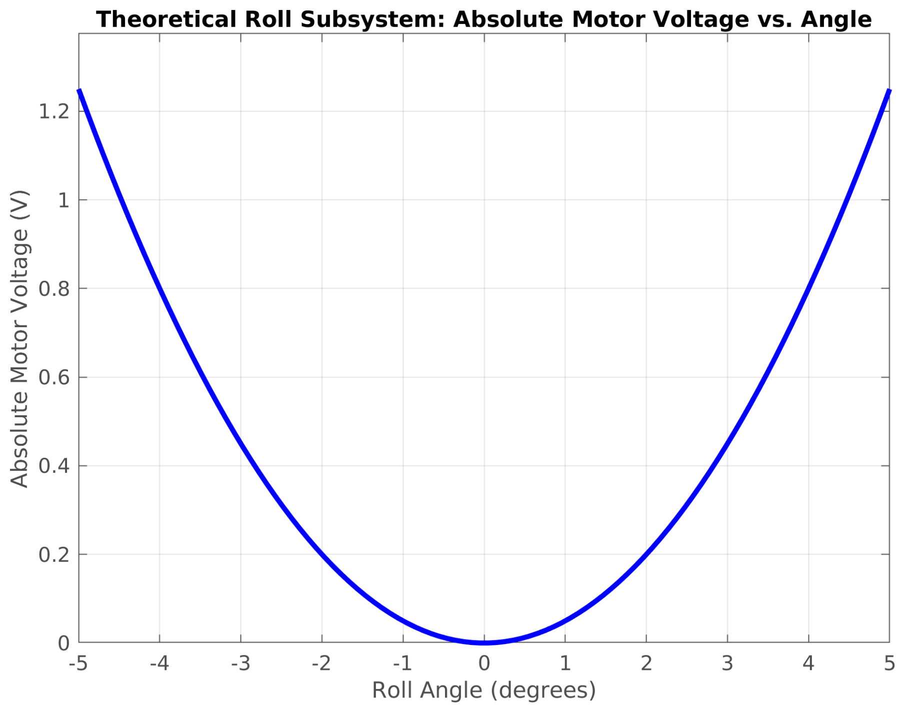

Figure 4.5: Theoretical trend of absolute motor voltage required for the roll subsystem as a function of roll angle.

For the pitch subsystem, the force required by the actuator actually decreases as the joint moves through its range of motion from -30 degrees to 30 degrees. The theoretical voltage required for the pitch subsystem across its range of motion is shown in Figure 4.6.

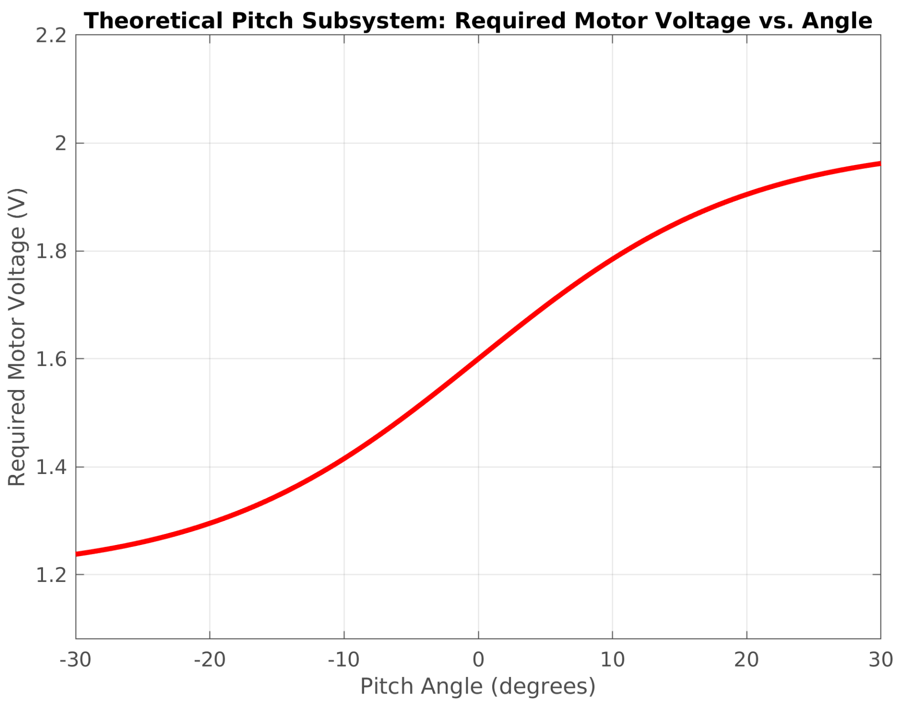

Figure 4.6: Theoretical trend of required motor voltage for the pitch subsystem as a function of pitch angle.

Additionally, complexities are added to the system dynamics when one realizes that the force required by the roll subsystem is also coupled to the pitch subsystem's position. As the pitch subsystem's angle is increased, the force required by the roll subsequently changes due to the system's center of mass changing position.

Similarly, the force required by the pitch subsystem varies with the position of the roll subsystem. As the roll subsystem diverges from the 0 point, the force required by the pitch subsystem lessens as it stays farther from its default position (normal to the gravitational force vector).

The combination of these factors necessitated the need for a more complex controller strategy in which the controller gains varied in accordance with the system position. The idea of this improved controller was that each joint's gains would vary with the current position of each subsystem.

This adaptive controller algorithm would require the characterization of the system at multiple points of the system's position. It was decided that 9 characterization points would be used for each joint of the subsystem. The pitch would be varied from $-15°$, $0°$, $15°$ and the roll subsystem would vary from $-20°$, $0°$, $20°$. This resulted in the following grid of 9 characterization points:

Once the necessary characterization points were established, the system needed to be identified at each of these points for each joint, resulting in the necessary system identification procedures. The actual system identification process was fairly straightforward. The previously established strategy for each joint was used; the only difference was that the positional setpoint for each characterization point varied to ensure the joint

Table 4.2: Grid of 9 Characterization Points for System Identification

	Roll Angle (°)		
Pitch Angle (°)	**-20**	**0**	**20**
-15	$(-15,-20)$	$(-15,0)$	$(-15,20)$
0	$(0,-20)$	$(0,0)$	$(0,20)$
15	$(15,-20)$	$(15,0)$	$(15,20)$

not being characterized remained at the necessary setpoint. The controller designed around the $(0,0)$ characterization point was used. While it was found that the dynamic performance of the controller designed around the $(0,0)$ setpoint degraded as the system diverged farther from the original linearization point, this controller performed well enough in the case of a static setpoint that it could be used to hold the extreme angles required by the multi-point system identification.

Once the necessary characterization points were established, the system needed to be identified at each of these points for each joint, resulting in the necessary system identification procedures. The actual system identification process was fairly straightforward. The previously established strategy for each joint was used; the only difference was that the positional setpoint for each characterization point varied to ensure the joint not being characterized remained at the necessary setpoint. The controller designed around the $(0,0)$ characterization point was used. While it was found that the dynamic performance of the controller designed around the $(0,0)$ setpoint degraded as the system diverged farther from the original linearization point, this controller performed well enough in the case of a static setpoint that it could be used to hold the extreme angles required by the multi-point system identification.

Upon the completion of each system identification trial, the resulting system response was saved in a CSV file under the following format: `IDENTIFIEDJOINT_ROLLANGLE_PITCHANGLE.csv`. A MATLAB script was then used to parse each of these files, identify the system at that point, and store and plot the system identification results.

4.3 Controller Theory and Design

The controller design was paramount to this research; a high emphasis was placed on system performance, and proper controller design would directly drive this. As a result, a large amount of effort was put into both the design and tuning of the controller.

4.3.1 Original PID Controller

The system originally utilized a very basic PID controller. Tuning was conducted with the Ziegler-Nichols method, and the following gains were found for each subsystem:

Table 4.3: Ziegler-Nichols PID Gains for Roll and Pitch Subsystems

Subsystem	k_p	k_i	k_d
Roll	35	50	5
Pitch	60	110	5

However, with these values, when full PID control was used, the system would rapidly vibrate, leading to uncontrolled system behavior and unacceptable noise and vibration. Because of this, the original system was only ever used with PD control, and its performance suffered greatly. Following sinusoidal trajectories, it would frequently undershoot or overshoot the setpoint and responded poorly to perturbations. As a result, an improved controller method needed to be developed.

4.3.2 Model Predictive Control (MPC)

Initially, several types of controllers were considered for this system. Model Predictive Control (MPC) was initially considered, as the system was already partially modeled in the form of an SDF model. MPC uses a dynamic model of the system to predict system behavior and determine a cost-optimal control input based on the model dynamics. While this control technique would result in extremely robust and optimal control, two factors prevented its utilization. First, the system model was a purely geometrical model, only accounting for the mass and inertia properties. Internal dynamics, such as frictional forces, were not considered, and the complexities of modeling such behavior were daunting. Secondly, MPC requires high computational power to simultaneously run and predict the model's behavior. This system utilized low-cost hardware with limited computational resources and, as a result, lacked the computational power to perform MPC.

4.3.3 Gain Scheduling and Adaptive Control

It was clear that due to varying system dynamics across the system's range of motion, adaptive control or gain scheduling would be a very valuable asset in improving system performance. Adaptive control involves online estimation of unknown or time-varying system parameters, or direct adjustment of controller parameters based on real-time performance feedback. In contrast, gain scheduling is a pre-programmed approach where controller parameters are adjusted based on a known, measurable "scheduling variable."

With relatively little additional computational power and complexity, the control gains could be calculated and interpolated based on the system's position, making gain scheduling an excellent candidate for improving system performance.

4.3.4 LQR and LQI Control

Linear Quadratic Regulator (LQR) control and Linear Quadratic Integral (LQI) control are cost-optimal control algorithms that minimize a quadratic cost function. LQI control extends the efforts of LQR control by incorporating integral action, which reduces steady-state error and improves system performance.

Both of these control methods are well-suited to this research as they achieve a good balance of high system performance with minimal control efforts. LQR control was chosen for the roll joint, as it was found that its performance was more than adequate for the system. However, during testing of the pitch subsystem, it was found that the system was consistently undershooting the desired positional setpoint. It was found that an LQI controller would minimize this steady-state error and drastically improve performance.

The objective of LQR control is to find a control input $\mathbf{u}(t)$ that minimizes the following quadratic cost function:

$$J_{LQR} = \int_0^\infty (\mathbf{x}^T(t)Q\mathbf{x}(t) + \mathbf{u}^T(t)R\mathbf{u}(t))\,dt$$

where:

- $\mathbf{x}(t)$ is the state vector.
- $\mathbf{u}(t)$ is the control input vector.
- Q is a positive semi-definite weighting matrix for the states, penalizing deviations from the desired state (often the origin).
- R is a positive definite weighting matrix for the control inputs, penalizing control effort.

For LQI control, the system is augmented with an integral of the error, and the cost function is similarly minimized. If $\mathbf{e}_I(t)$ represents the integral of the output error, the augmented state $\mathbf{x}_{aug}(t)$ includes the original states and the integral error. The cost function for LQI is then:

$$J_{LQI} = \int_0^\infty (\mathbf{x}_{aug}^T(t)\tilde{Q}\mathbf{x}_{aug}(t) + \mathbf{u}^T(t)R\mathbf{u}(t))\,dt$$

where $\tilde{Q}$ is the augmented weighting matrix for the states, including a penalty on the integral error to drive it to zero.

4.3.5 Feedforward Control

Feedforward control is a control method which attempts to anticipate changes or disturbances to the system's input. It was theorized that a feedforward control strategy could further reduce any remaining steady-state error in the pitch subsystem.

In order for feedforward control to be implemented, a model of the system must be constructed in the form of a specific matrix known as the $B^\dagger$ matrix. The $B^\dagger$ matrix is formulated through the Moore-Penrose pseudoinverse. It is formulated as follows:

Let B be an $m \times n$ real or complex matrix. Its Singular Value Decomposition (SVD) is given by:

$$B = U\Sigma V^T$$

where:

- U is an $m \times m$ unitary (or orthogonal for real matrices) matrix whose columns are the left singular vectors of B.
- Σ is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, called the singular values of B, typically arranged in decreasing order.
- V^T is an $n \times n$ unitary (or orthogonal for real matrices) matrix whose rows are the right singular vectors of B. (V is the matrix of right singular vectors).

The **Moore-Penrose pseudoinverse** of B, denoted $B^\dagger$, is then defined as:

$$B^\dagger = V\Sigma^\dagger U^T$$

where $\Sigma^\dagger$ is an $n \times m$ matrix formed by taking the reciprocal of each non-zero singular value on the diagonal of Σ, and then transposing the matrix.

The pseudoinverse $B^\dagger$ is unique and provides a "least squares" solution to linear equations. For a system $Bx = y$, if an exact solution doesn't exist, $x = B^\dagger y$ provides the solution that minimizes $||Bx - y||_2$.

For the purpose of this research, this matrix could be calculated via the MATLAB command `pinv`. This command, when applied to the identified B matrix from the system identification procedure, would result in the $B^\dagger$ matrix.

This procedure was conducted for each of the characterization points previously discussed. Upon calculation of the $B^\dagger$ matrix, this matrix is used to calculate u_{ff} by multiplying the $B^\dagger$ matrix (pseudoinverse of the input matrix) with the sum of the system's A matrix multiplied by the reference state and the time derivative of the reference state. This u_{ff} is then scaled by a factor λ, which can be tuned from 0 to 1.0 to adjust the feedforward element of the controller.

The ultimate result of this control effort is to proactively cancel anticipated system dynamics and generate better system performance.

4.3.6 Comprehensive Control Strategy

The final controller for the system was a resulting combination of several of the control strategies previously discussed. Ultimately, for the roll joint, it was found that gain scheduling with an LQR controller provided adequate system performance. However, for the pitch subsystem, this method needed to be improved upon due to consistent steady-state error in the form of target undershooting. To remedy this, an LQI controller with feedforward implementation and gain scheduling was employed. These strategies led to drastic improvements in system performance.

5 System evaluation

5.1 Overview of Experiments

5.2 Trajectory Tracking Performance

5.3 Effect of Adaptive Control

5.4 Summary of Results

6 Conclusion and Future Work

6.1 Summary of Research Objectives

6.2 Key Findings

6.3 Limitations

6.4 Recommendations for Future Work

6.5 Final Remarks

Bibliography

[1] dVRK Community, “The da vinci research kit,” https://github.com/jhu-dvrk/dvrk-ros, 2023.

[2] Author19, “Title19,” 2019.

[3] Author20, “Title20,” 2020.

List of Figures

List of Tables

List of Source Codes