Design of a Portable Teleoperation Interface with Haptic Feedback for the Mirte Master

Aditya Swami, Duncan Bijkerk, Ynze Visser, and Quita Carriere

June 2025

Abstract

This paper presents a teleoperation interface for the Mirte Master. The teleoperation interface allows for 0th-order control and offers force feedback for the gripper. Additionally, the design is a non-wearable portable device that ensures user mobility during teleoperation. The design incorporates a master-slave control system for manipulating the arm of the Mirte Master, for which the input is a scaled-down version of the Mirte Master´s arm. A joystick controls the movement of the base. A pinch mechanism, with haptic feedback provided by a series elastic actuator, controls the gripper. A compact casing with a shoulder strap ensures the controller is portable. The controller is capable of teleoperating the Mirte Master to execute the needed pick and place actions. The force feedback system is unstable. The experiment results indicate an average communication latency of around 0.4 seconds and positional deviation up to 0.3 radians from the input commands from the controller.

Figure 1: *Design of the teleoperation interface, as seen from the perspective of the operator*

1 Introduction

The Dutch horticultural sector is one of the biggest economic sectors of the Netherlands. In 2021 alone, the total production value of the horticultural sector amounted to 31.2 billion euros [1]. It is at the forefront of innovation, pioneering efficient and sustainable technologies. Unfortunately, the harvesting process usually involves a lot of unstructured, repetitive manual labor.
The concept of teleoperation enables humans to stay in control whilst relegating the physical labor to a robotic manipulator. To make teleoperation possible, a remote controller interface that the operator can manage is needed.

Whilst fruit is ripening, the stiffness continues throughout. The stiffness values of fruits in the pre-harvest, harvest order, and table periods differ [2]. To be able to pick fruit at the right moment, a distinction needs to be made between the different fruits and their stiffness. Touch affords precise control and discrimination and allows the assessment of an object's dynamic and material properties [3]. Therefore, haptic feedback is a central design requirement for the gripper control to ensure a precise feeling of the stiffness of the picked fruit.
When operating a mobile manipulator, the operator needs to be able to observe the state of the robot. To enable the operator the needed vision of the mobile manipulator, portability of the controller was determined as a key design aspect. The controller has a max weight of 1.5 kilograms and max dimensions of 275 x 150 x 100 mm to ensure the ease of movement of the controller.
The mobile manipulator, that the controller is designed for, is the Mirte Master (MM), shown in Figure 2. The MM is a mobile robot with a manipulator arm with a gripper as end effector. The MM was designed as an educational robot, and will be used in greenhouses to har-

Figure 2: *The Mirte Master robot, the educational robot for which the teleoperation device is built for*

vest tomatoes for experimental purposes.

One of the previously designed controllers for similar application is the Kraft KMC 770, as seen in fig. 3a, referenced in [4]. This is a master controller for a series of robotic arms developed by Kraft. The external manipulator arm can be controlled intuitively by the miniature arm located on the KMC 770. The miniature arm also provides feedback through actuators located in each of the joints. However, the drawbacks are its size and weight, measuring 40 cm by 20 cm and weighing 5.2 kilograms, it lacks portability.
Alternatively, the Tel-Punto by ELCA, as seen in fig. 3b, is portable [5]. This device is strapped to the waist. It weighs just under a kilogram and is also quite small, at 175 x 140 x 132 mm. Nevertheless the operation is done by two joysticks, which can be unintuitive and impractical for manipulating the arm on the MM. Furthermore, it has no ability to provide any force feedback for the operation.
The Force Dimension delta 3, as seen in 3c, is designed to provide haptic feedback. The configuration of the Force Dimension Delta 3 allows the force feedback motors to be mounted on the base. This causes the motors to remain stationary during operation, minimizing the effects of the inertia of the moving parts. As a result, the system can accelerate faster and can thus handle a wider range of frequencies, achieving a higher bandwidth [6]. However, the teleoperation device for the MM won't have haptic feedback for arm movement, making this advantage irrelevant. Moreover, this configuration with the three arms occupies a large amount of space relative to the available end-effector workspace, which would limit the portability of the device.

The existing controllers for similar applications have varying limitations in portability, force feedback, and intuitive control. The current design aims to develop a teleoperation interface for the Mirte Master that would retain the main advantages:
- Haptic feedback, provided in the gripper;
- Intuitive design for operation, by offering 0th order control;
- Portable design, meaning limited size and weight, and being non-wearable.
This paper will focus on the design choices made regarding the integration of the hardware and software. For the design process see Appendix H.
The functionalities of the controller will be explained in the 'Design of the Teleoperation Interface' section. Then the 'Experiments' section will first establish certain tests and subsequently look at the results. Finally, this paper will finish with a 'Discussion and Conclusion', in which a reflection on the design is discussed.
This design study has been conducted as part of the Bachelor's End Project for Mechanical Engineering students at the Delft University of Technology.

(a) KMC 770

(b) Elco Tel-Punto

(c) Force Dimension delta 3

Figure 3: *Existing Technologies used for Inspiration*

2 Design of the Teleoperation Interface

The design of the controller, see Figure 4, consists of three main control components: the controller arm (red) for manipulating the MM's

robotic arm, a pincher (magenta) for operating its gripper , and a joystick (blue) for driving the base. Toggle switches (green) are also available to; toggle between holonomic and nonholonomic drive modes; mirroring the arm; and power. The joystick and toggle switches are off-shelf parts and were chosen for their familiarity. The design of the arm and the pincher will be discussed in detail below. The design is integrated with electronics and software.

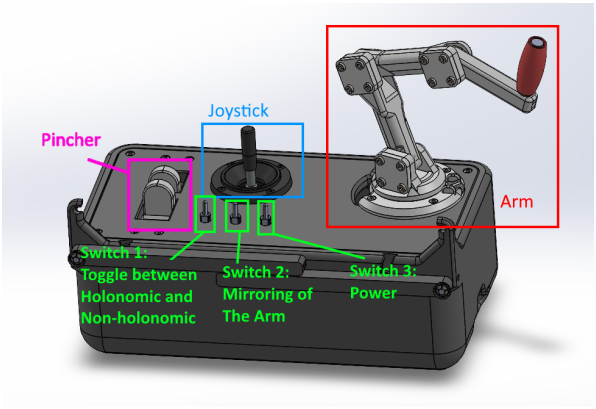


Figure 4: *Render of the teleoperation device with the major components: Joystick, Pincher, and Arm used for the movement of the base, the gripper. and manipulator arm respectively on the MM.*

2.1 Controller Arm

The controller arm is approximately a 2:1 scaled-down version of the MM's arm, which features four degrees of freedom. It is manipulated by moving a freely rotating handle located at the end-effector through 3D space. The joint angles on the controller arm are measured using rotary magnetic encoders. The MM's arm is controlled through master-slave control, which allows the operator to move the MM's arm fluidly and intuitively.

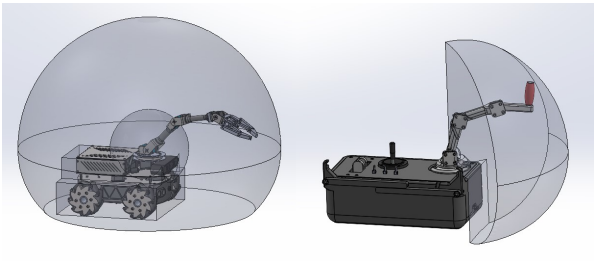


Figure 5: *Workspaces of the MM arm and the controller arm. The controller arm does not encompass the entire range of the MM, mainly due to differences in the range of the yaw and the elbow joints.*

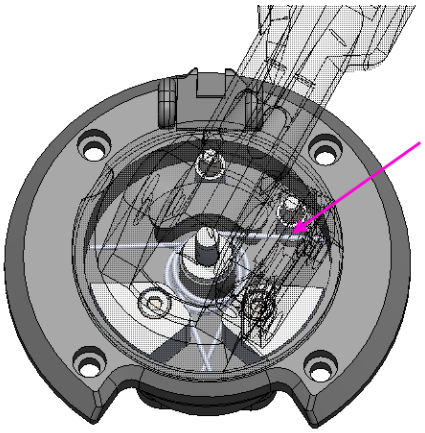


Figure 6: *View of the first order control springs in the yaw axis. Here the Yaw joint is positioned at the boundary between 0^{th} and 1^{st} order control. The left bolt, which is connected to the turret, can be seen making contact with the pre-tensioned torsion spring.*

2.1.1 Workspace Mapping

A comparison of the two workspaces is presented in Figure 5. A direct replica of the MM's arm workspace on the controller would be impractical due to ergonomic limitations. Therefore, the yaw axis of the controller arm is limited to 100° of rotation. The yaw axis of the MM itself supports a range of 180°.
To enable control across this full range, a hybrid control strategy is implemented on the controller arm, see the code in Appendix E. The first ±30° of motion from the center operates in 0^{th}-order control mode, where the arm directly maps position. Beyond this range, 1^{st}-order control is engaged, allowing for a velocity-based input. A pre-loaded torsion spring applies a push-back force in the velocity control

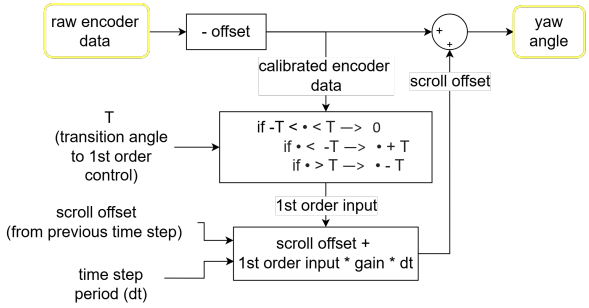


Figure 7: *Block diagram of the yaw, observe that the raw encoder data has to be calibrated and transformed to* 1^{st} *order for the scroll function if needed.*

range, see figure 6. This helps the user identify when they have moved beyond the 0^{th} order control zone.

The 1st order scroll-offset is determined by the deviation of the yaw axis past the boundary angle, see Figure 7 for the block diagram. First, the yaw of the controller arm is calibrated for the encoder offset. If the controller's yaw is within the 0th order range, no change is made to the scroll offset, but if the spring is engaged, the level of compression of the spring is factored with the time step period (dt) of the control loop and the gain, and added to the scroll offset of the previous control step.

The MM's yaw axis only has a range of 180°, the remaining half of the workspace can be accessed by flipping the arm to the opposite side. However, the controller arm alone only partially supports this opposite-side workspace, and it lies outside the ergonomic control range. To enable complete access to this region, a switch has been added that, when activated, scales all joint angles (except for the yaw joint) by -1, effectively mirroring the arm's configuration.

2.1.2 Elbow Deadzone

With a passive arm of this configuration, when the arm is placed in its extended position, gravity causes the elbow joint to drop into a low, "elbow-down" posture. This effect is caused by a dead-zone in the elbow joint, where the passive mechanics cannot control the elbow angle. In this state, the user cannot return the arm to a preferable "elbow-up" configuration using only the end-effector knob. This issue is solved by mechanically limiting the range of the elbow joint so that it cannot overextend. The minimum end-stop angle was found to be around 10 degrees. This ensures that retracting the arm from its most extended position doesn't require a significantly higher force compared to manipulating the arm in other positions. Limiting the range of the elbow does mean that the maximum extension of the arm is reduced by 1.5%, which is deemed an acceptable loss compared to the improvement in control.

2.2 Pincher

The pincher mechanism, see Figure 8 controls the gripper of the MM and provides force feedback to the user. A symmetrical linear pinch design was chosen to mimic the movement of gripping an item. This linear motion is converted into rotational motion via a rack-and-pinion, allowing the use of a rotary actuator and a rotary magnetic encoder.

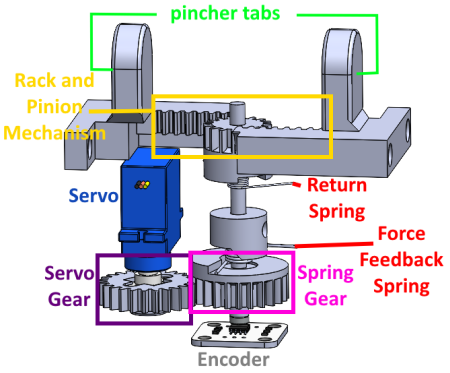


Figure 8: *Stripped CAD model of the pincher mechanism. The operator pinches at the pincher tabs making the rack and pinion turn. The servo turns the servo gear to connect the spring gear and force feedback spring to enable force feedback.*

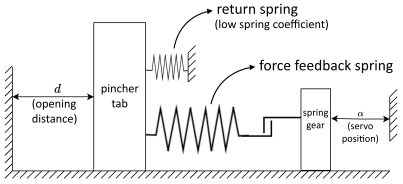


Figure 9: *Schematic of the pincher mechanics, including the SEA. When the servo turns the spring gear, the force feedback spring pushes back onto the pincher tabs creating force feedback.*

2.2.1 Force Feedback Actuator

The teleoperation device is intended to be used for harvesting tomatoes. This means that the objects that will be grasped will all have a relatively similar stiffness. A series elastic actuator (SEA) used for generating force feedback suits this scenario well because the elastic element can be tuned to be approximately the same stiffness as a tomato, enabling high fidelity force feedback [7]. The stiffness of a tomato lies in the range of 1 to 3 N/mm [8], so the stiffness of the selected torsion spring is 2.08 Nmm/deg, which translates to 1.9 N/mm at the pincher tabs.
Choosing to use an SEA over a direct drive actuator allows the use of a non-backdrivable motor, resulting in a smaller and lighter geared servo motor. This is especially important due to the limited space within the controller casing and the weight constraints of the device.
In addition to the force feedback spring, a return spring with a very low stiffness is added to make the pincher follows the user's fingers when they open. A schematic of the mechanics of the pincher mechanism, including the SAE, is presented in Figure 9.

2.2.2 Force Data Simulation

Getting usable force data from the MM's gripper servo was unsuccessful. A force- or current sensor could be added to the MM to retrieve this data. This is outside the scope of this project. Therefore a script that simulates force data from the gripper was implemented to test the performance of the force feedback on the pincher. This script models an object with a certain size and stiffness and returns a force depending on the input angle of the gripper. See Appendix F for the code.

2.2.3 closed-loop pincher system

A block diagram of the closed-loop system, including the Force Data Simulator (Section 2.2.2), is shown in Figure 10. Physical testing has demonstrated that this system is unstable, particularly near the boundary of contact with an object. One possible cause of this instability is the additional delay introduced by the servo taking time to make contact with the torsion spring. This issue could be mitigated by continuously keeping the servo in contact with the spring. However, doing so would require the servo to react more quickly than it currently can, in order to avoid affecting the movement of the gripper when it shouldn't. To improve stability, a PID controller can be added to the

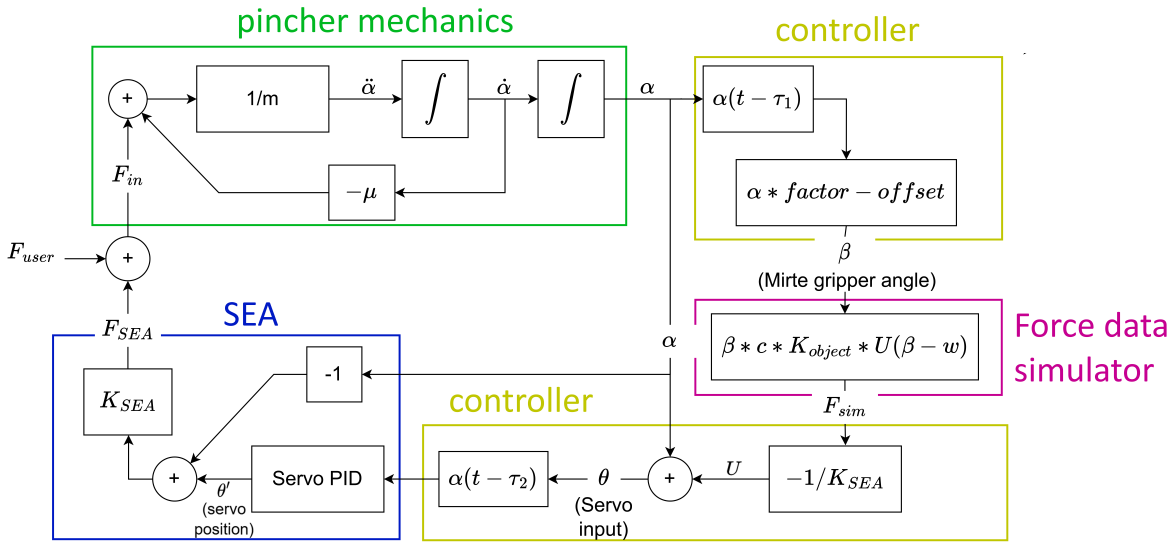


Figure 10: *The closed-loop pincher system with the force simulator. This system is proven to be unstable through physical testing. A PID controller should be added to the system, to make it stable. here α is the raw encoder value from the pincher, μ is the friction factor from the pincher machanism, U is the desired spring depression and from the force simulator: c is a scaling constant, K_{object} is the stiffness of the simulated object and w is the width of that object.*

system. This PID loop would take the error of the spring depression as input, which is the difference between U (the wanted spring depression) and $\theta' - \alpha$ (the actual spring depression). This would require an extra encoder to measure the servo angle, see Appendix C

2.3 Electronics

The 'brains and spine' of the controller were an Orange Pi and a Raspberry Pi Pico, also called Pico. The Orange Pi is used for communication between the controller and the MM., and has WiFi capabilities, meaning the controller can be portable and be used on any WiFi network. The Pico is connected to all actuators and sensors of the controller. To measure the angles in the controller arm, four encoders are needed, one for each DOF. One more encoder is used for the pincher mechanism, which measures how much it's opened/ closed.
The AS5600 Magnetic encoder was picked for being small, cheap, and accurate enough for the current application. These encoders use I2C and all share the same address, so they cannot be connected directly to the Pico's I2C bus. A multiplexer (mux) in between the Pico and encoders solves this by allowing multiple devices with the same address to share one I2C bus by use of separate channels. The joystick used is the JH-D202X-R4. This joystick was chosen because it is cheap and it has two simple analog outputs, one for the X and Y directions.
Three switches are also integrated into the design these are also off the shelf components. The servo motor used is the SG90 mini servo. It's a small and cheap servo and provides enough torque for the current application. It draws its power from the power bank directly because the Pico does not supply the servo with enough current. The electrical diagram is shown in Appendix B in Figure 18.

2.4 Software

The software is divided into three parts. See Appendix B Figure 17 for the diagram. The main part is the **Controller** code, written out in Appendix D. In this code the control loop is set up and executed, with a data collection at the end of the control period. To make the Controller code work, a script with **Classes** was used, see Appendix E. In the Classes the sensor data is collected and transformed into the values needed for the Control loop. Because the hardware of the MM does not allow for force sensing, a **Simulator** script for the force was used instead, see Appendix F.

2.4.1 Control Loop

In the control loop of the Controller script, the teleoperation is regulated. During the while-loop, all sensor data is read out through the classes.
First, the switch data is read out every 0.5 seconds. This is due to the switch data call being a ROS Service, and services needing more time to receive a response, thereby slowing the control process down. The first switch is used to change the driving style between holonomic and angular driving. The second switch is used to change the direction of the arm movement. Both switch functions allow for a short wait period for the change to occur. Therefore, the delay of up to 0.5 seconds is considered allowable to speed up the general control of the MM.
The joystick output is calibrated during the first run through of the control loop and sets the neutral position of the joystick. After the calibration, the intensity of the joystick output is mapped proportionally to the speed of the MM. This is capped at the lower end, so as not to have the motors stall when very low input is given to the joystick.
The controller arm output is the angle between the two respective controller arm parts. Due to the kinematic mimicry of the controller arm, the angles are mapped directly from the controller to the MM, only changed by the offset of the magnetic encoder.
The yaw angle is incorporated in the controller arm output, but has the added feature of a shifting scroll-offset due to workspace limitations. For more information on the yaw scroll-offset, see section 2.1.1.
The pincher output is not directly proportional with the gripper angle, as the encoder measures the rotation of the gear in the pincher system. This measurement is translated through an offset and a scaling to get the needed gripper angle.
Due to the MM not having a force sensor, the force needed for the force feedback is defined in the gripper simulator, see section 2.2.2 The needed servo angle in the pincher for the calculated force feedback is determined

based on the angular position of the gear in the pincher and the spring constant of the torsional spring.

All these values are published to the MM, causing it to move, and the loop starts over.

2.4.2 Frequency and Duration of the Controller Loop

In the Control loop, a noticeable delay was observed. To analyze the cause, a delay measurement was done, see Figure 11. Here

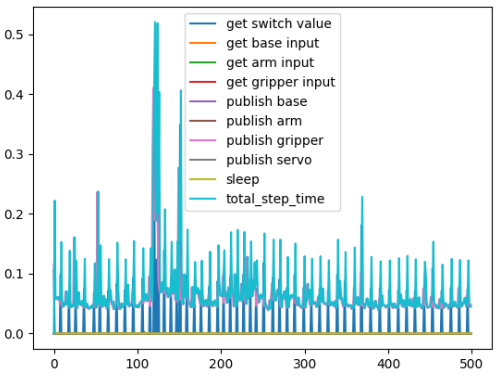


Figure 11: *Original code, without delay effect mitigated. The actions of 'publish gripper', and 'publish servo' contribute most to the total step time.*

it is visible that the 'get switch value' and 'publish gripper' actions take longer than the other actions. These two are ROS Services, unlike the rest, which are ROS Topics. When a service call is initiated, a call message is sent out, the connection between ROS nodes is made, and a callback is sent back. A topic, on the other hand, consists of only the sendout of a message. To mitigate some of this effect, the service calls were changed with the 'persistent' argument, which keeps the nodes connected [9].
Keeping the client connected to the service had a strong effect on the loop speed, see Figure 12. The average time for these actions went from around 0.1 to below 0.025 seconds.

In the yaw scroll-offset, discussed in section 2.1.1, the scroll-offset is determined every step. The shortening of the time per loop does not affect the speed of change of the scroll-offset in the yaw direction, as the duration of the time step is taken into the calculation of the

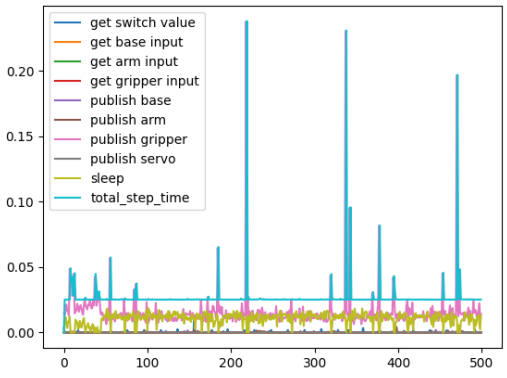


Figure 12: *When delay mitigation is applied with 'persistent' service connection, the time one loop takes decreases.*

scroll-offset change.

2.4.3 Calibration

A calibration cycle is part of the setup. The current script, see Appendix D, has the calibration turned off, and the needed values defined in the setup. When the controller is used for the first time in a while, the calibration can be turned on, so the offset of different axles is gathered automatically, thereby making the use of the controller more foolproof. To calibrate the controller, the arm is placed into various set positions, and encoder data is read out and used as the offset.

3 Experiments

The following experiments are done to determine whether the controller works as intended and identify possible areas of improvement.

3.1 Experiment Design

The experiments that have been preformed are:

1. Pick and Place: The tasks the robot should perform include pick and place actions. To show the integrated working of the controller, a simple pick and place action will be performed, and the position of the end of the arm was collected. The movements needed are: Go from full upward extension to the position of the object; Pick up the object; Turn to the other

side of the robot; Place the object; Return to full upward extension.

2. Latency test: Measure the response delay between user input on the controller and the initiation of movement in the MM's joints. This measurement provides information about the overall responsiveness and speed of the system. The delay information is gathered from the command and servo data that is gathered during the running of the MM. Within this data, the biggest delay will be visually determined and subsequently gathered.

3. Joint deviation test: The objective of the joint deviation test was to asses how the servo angles in each of the MM's joints deviate compared to their respective angles in the controller arm during manipulation. To perform this test, each joint will be operated individually over a predetermined period of time. Data is collected via software during the test. Servo data is then compared to the encoder data.

3.2 Experiment Results

This subchapter will describe the results from the experiments and what can be concluded from them.

1. Pick and Place: The pick and place task shows that the controller is capable of the necessary movements to perform its intended task. The position of the end of the arm is steady, see Figure 13, and the task is executed correctly. Snapshots of the Mirte Master can be seen in Figure Appendix A, these positions correspond with the time points accentuated in Figure13.

2. Latency test: The latency test shows that all the joints experience some level of latency. As seen in Figure 14 for the wrist, the delay is not steady for all moments. The delays of all controls stay under 0.8 seconds at all times. Bigger delays than 0.4 seconds are seen as outliers. For the wrist and pincher, the latency is the least compared to the other joints. The closer the joint is to the MM, the more latency it experiences. This is likely because of inertia, as they move more mass. Due to the time delay, the operator cannot immediately see the result of their input, which can lead to over- or undershooting and increased difficulty with fine manipulation of the arm. The severity is dependent on the amount of latency.

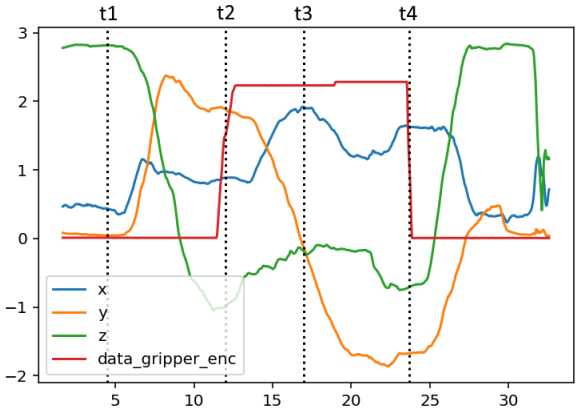


Figure 13: *This graph shows the gripper x/y/z-position from the pick and place test over time.*

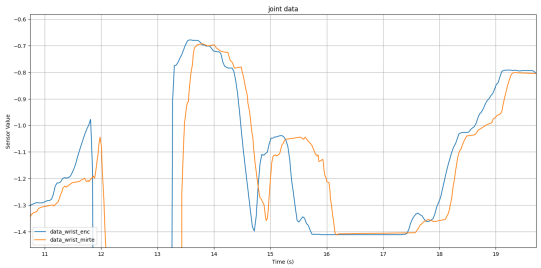


Figure 14: *Zoom in of the wrist delay, around second 14.5 delay is visibly shorter than around second 15.8.*

3. Joint deviation test: Figure 15 shows the deviation in all the joints for a predetermined period of time.

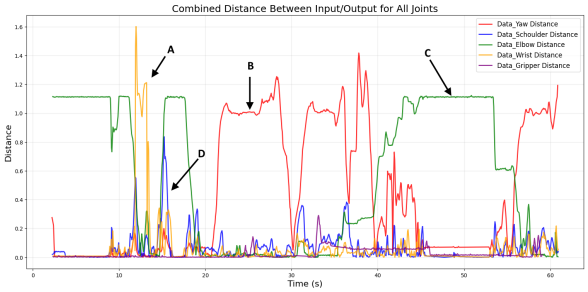


Figure 15: *The deviation between the input/output for all joints over time, Excluding the outliers shows the average deviation stays below 0.4 radians. There are some exceptions which are labeled.*

It becomes evident that the system demonstrates reliable accuracy, with an average deviation of approximately 0.3 radians, excluding the outliers labeled A to D. The graph is based on the data points, which are shown in fig. Appendix G. The outlier, which is labeled A in the graph, shows a sudden big difference at the beginning between the wrist input/output, which is likely due to the start-up procedure of the MM. Outlier B shows that the yaw data frequently deviates greatly. The points in time where this happens are where the scroll offset is used on the controller, the offset corresponds with the scroll offset. Outlier C is since the controller elbow joint has a larger range of motion than the MM elbow joint, and can thus sends data that the MM cannot process. The final outlier D is due to the delay. Because the shoulder distance has a sharp ascent and descent, it may seem like the deviation is large. The sharp ascent and descent are caused by the latency of the system and are only visible on the graph, making it a visual inconsistency. The large deviation does not occur in reality. Excluding the outliers shows that the input and output align pretty closely, showing an accuracy of 0.3 radians.

4 Conclusion and Discussion

We have developed an interface for an agricultural robot where manipulation is done through mimicry, mobility is operated through a joystick, and our teleoperation interface successfully combines 0th-order control with haptic feedback for the gripper. These key features allow for intuitive and fine manipulation of the MM. The novel combination of these factors is what distinguishes it from existing technologies. The compact design allows for portability. The size and weight are 256x136x88 mm and 1.13 kg, respectively, satisfying the design requirements.

This design was experimentally evaluated. The results of the deviation and latency test show: excluding the outliers, the average deviation between the input/output stays below 0.3 radians, and the latency is roughly 0.4 seconds. The pick and place proved that arm manipulation can be done precisely enough for the current application. These results together show that the gross movement needed for horticultural means can be executed reliably.

Reflecting on our design allows us to differentiate some key differences and advantages from existing technologies. Our compact and portable design is what sets it apart from designs like the KMC 770 and the Force Dimension Delta 3, where the operators remain stationary when manipulating the controller arm. There are existing teleoperation interfaces that offer portability like the Elco Tel-Punto, but these designs don't allow 0th-order arm manipulation and haptic feedback, whereas ours does.

Furthermore the accuracy of the controller arm comes out to 0.3 radians as mentioned in section 3.2. This allows 0th-order control to be performed accurately. Additionally, the loop frequency has also been improved, as mentioned in section 2.4.2, to 0.25 seconds.

As stated before the MM is designed as an educational robot and will be used in greenhouses in an experimental setting to harvest tomatoes. Our teleoperation interface will aid in this process and shift the physical labor of the harvesting process to the robot, while still ensuring a natural and intuitive experience because of the haptic feedback of the gripper and the 0th-order control.

When operating, the yaw of the Mirte Mas-

ter is quite shaky, due to the encoder in the controller transmitting unsteady data. The true underlying cause of this fluctuating data could not be resolved, but it is possibly caused by electromagnetic interference, as the encoder and cables of the yaw axis are closest to the main electronic hub and its associated data cables. This theory should be tested and fixed for further development of the controller.

Regarding the force feedback, the main limiting factor on this pincher design is friction within the slider mechanism. While the slider moves smoothly when unloaded, it tends to bind when the actuator applies force. Due to this friction, the force from the actuator is not accurately transmitted to the user's fingers, drastically reducing the accuracy of haptic feedback. Replacing the plain bearings with linear ball bearings would significantly reduce friction and improve responsiveness, due to the lower friction. If this change proves insufficient, modifying the slider geometry to minimize binding may offer further improvement.

The design has some smaller improvements that have not been implemented due to time constraints. These are listed below.

- In the hardware of both the MM and the controller, a strong microcomputer was used for computing. This is excessive for the amount and type of calculations and computing that are needed for the current operation. For cost and material efficiency, an adjusted setup could be developed that relies solely on the microcomputer that is present in the MM.

- The elbow deadzone discussed in section 2.1.2 currently limits the effective workspace of the controller. This can be circumvented by flipping the movement direction and moving the controller arm upside down. This method is awkward and unintuitive. To make the 'dropped elbow' workspace reachable, a switch could be added that mirrors the elbow over the shoulder-wrist line. Thereby keeping the position of the gripper at the same point, whilst allowing the elbow to drop.

Acknowledgments

The authors thank the following individuals for their specific contributions: Prof.dr.ir. M. Wisse, J.M. Prendergast, and L. Peternel for their supervision and insights during this project. Arend-Jan van Hilten and Jasper van Brakel for their continuous support, both whenever we got stuck, as well as when everything turned out right. We would also like to thank all the members of the Cognitive Robotics department, who came with such excitement when passing our table and seeing the project develop.

References

[1] C. B. voor de Statistiek, "Tuinbouwcijfers 2022," 2023.

[2] H. Üstünel, E. Serdar Güner, M. O. Özcan, and B. Aslan, "Force feedback simulation design for fruit hardness," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, 2020, pp. 1–4.

[3] K. MacLean, "Designing with haptic feedback," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, 2000, pp. 783–788 vol.1.

[4] "Kraft Telerobotics - RAPtor Remotely Operated Force Feedback Manipulator Arm System." [Online]. Available: http://krafttelerobotics.com/products/raptor.htm

[5] "ELCA Radiocontrols - Waist portable Radiocontrol - CCS PUNTO." [Online]. Available: https://www.elcaradio.com/en/products/26-cableless_controls/8-punto

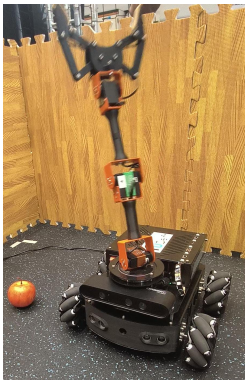
[6] K. Kemper, D. Koepl, and J. Hurst, "Optimal passive dynamics for torque/force control," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 2149–2154.

[7] Serie elastic actuators. [Online]. Available: https://mime.engineering.oregonstate.edu/research/drl/_documents/hurst_2020.pdf

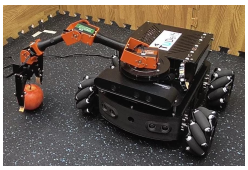
[8] R. Manivel-Chávez, M. G. Garnica Romo, G. Arroyo-Correa, and J. Aranda-Sanchez, "Optical and mechanical nondestructive tests for measuring tomato fruit firmness," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 8011, pp. 244–, 08 2011.

[9] roscpp/overview/services - ROS wiki. [Online]. Available: http://wiki.ros.org/roscpp/Overview/Services

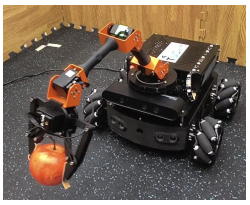
Appendix A Snapshots of Pick and Place



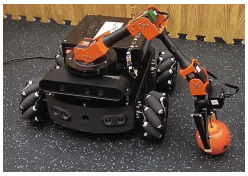
(a) *The arm straight up, corresponding to time stamp t1.*



(b) *The arm in pick position, corresponding to time stamp t2.*



(c) *The arm whilst moving to the other side, corresponding to time stamp t3*



(d) *The arm in place position, corresponding to time stamp t4*

Figure 16: *Snapshots of the pick and place actions.*

Appendix B Software and Electrical Diagram

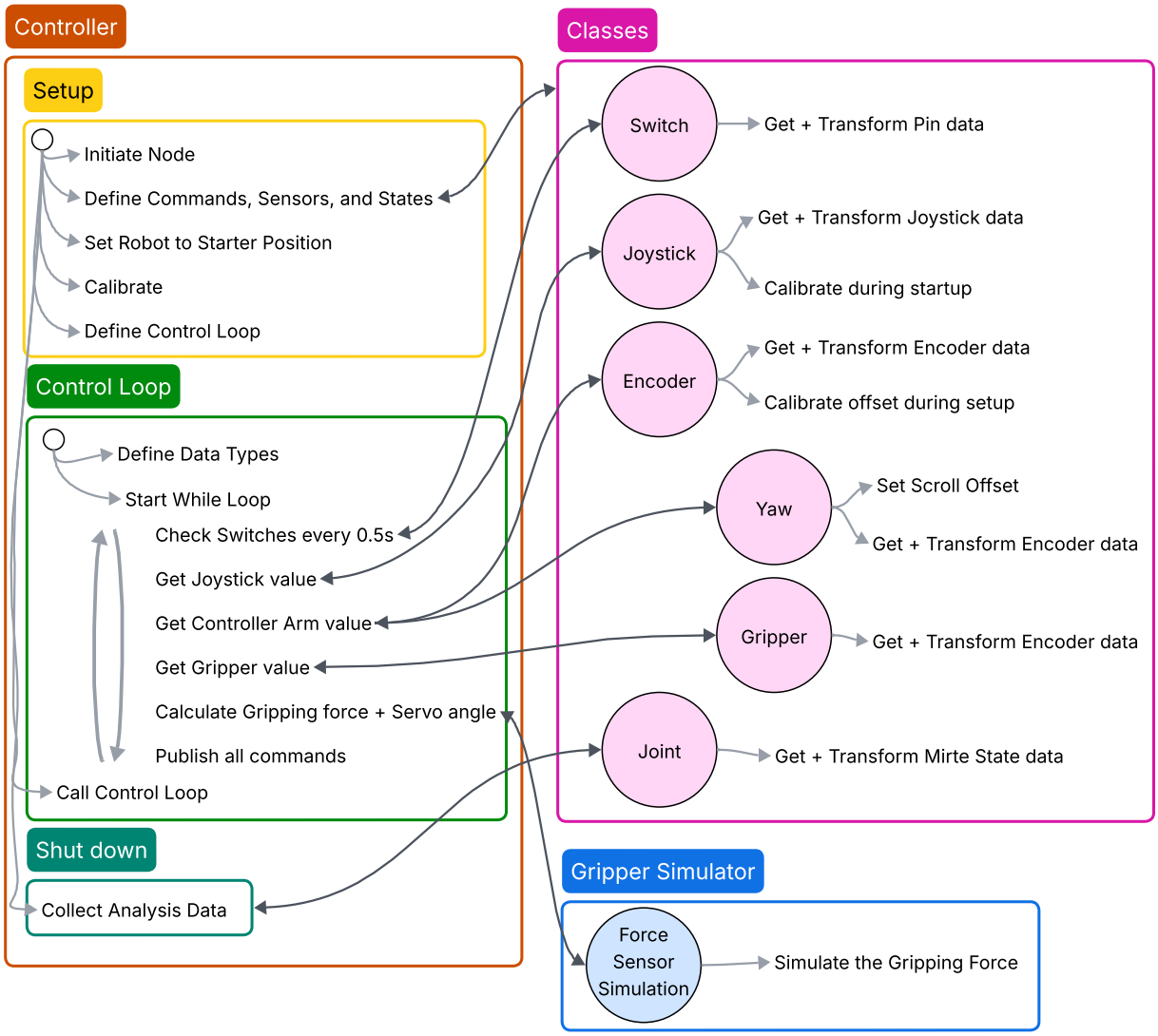


Figure 17: Diagram of the working of the software.

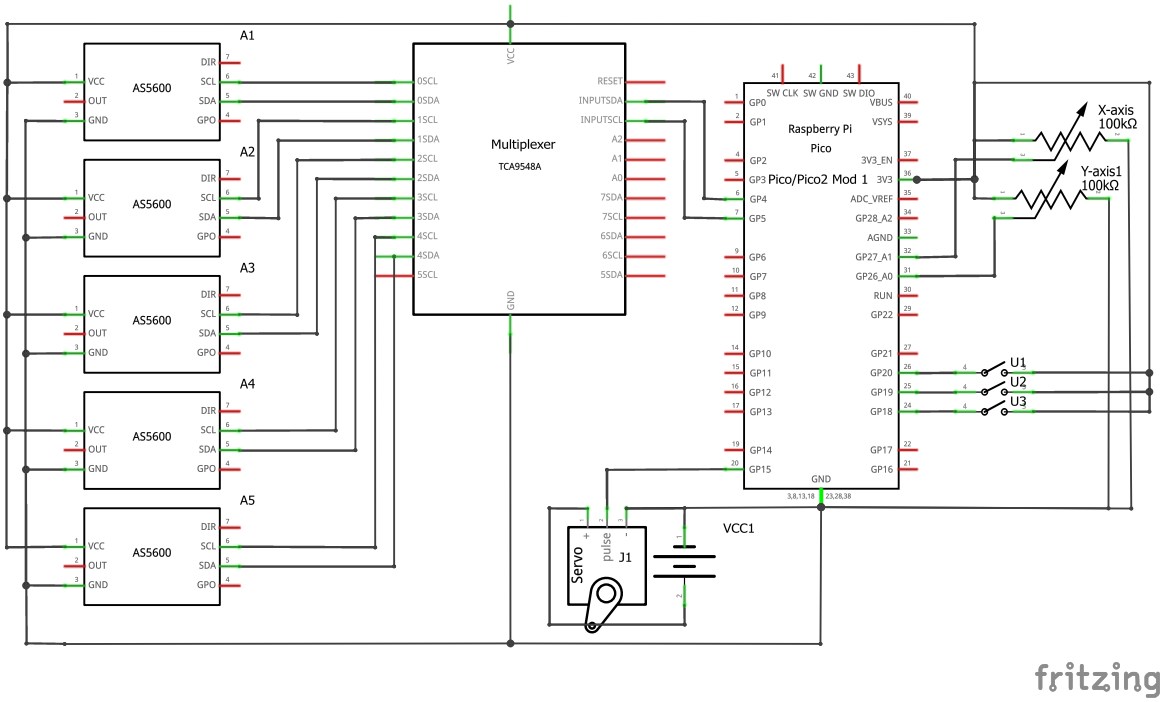


Figure 18: Electrical Diagram of the controller

Appendix C Closed Loop Controller Diagram with PID

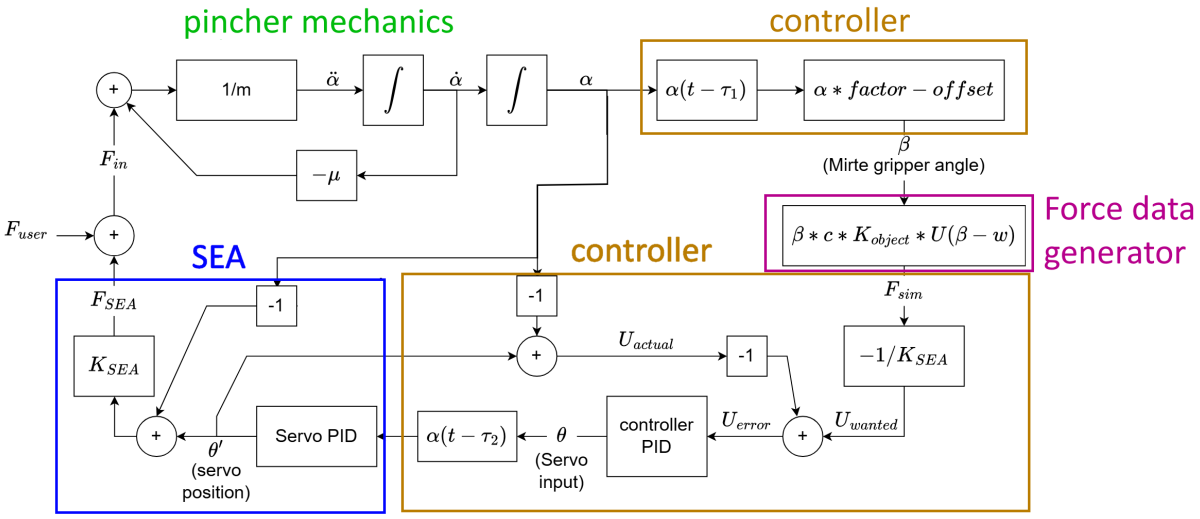


Figure 19: Closed Loop Controller Diagram with PID

Appendix D Main controller code, references classes and simulation

```
1
2  # ---- Import ----
3  import rospy
4  import numpy as np
5  from sensor_classes import Encoder, Joystick, Switch, YawEncoder, GripperEncoder, Joint
6  from gripper_simulator_file import GripperSimulator
7  import math
8  import csv
9  import pandas as pd
10 import time
11 # import data_collection_script
12
13 # load the specific message format required for joint commands
14 from std_msgs.msg import Float64MultiArray, Float32
15 from geometry_msgs.msg import Twist
16 from mirte_msgs.srv import SetServoAngle, GetPinValue, SetPinValue
17 from mirte_msgs.msg import ServoPosition, Intensity
18 from sensor_msgs.msg import JointState
19
20
21 # ---- Initialize controller node ----------
22
23 # start a new ROS node
24 rospy.init_node('simple_controller_publisher')
25 # provide on-screen information
26 rospy.loginfo('The base_controller_publisher has started!')
27
28
29
30 # ---- Set the publishers and services to control the position of the Mirte Master ----------
31
32 # create the publisher, tell it which topic to publish on
33 base_command_publisher = rospy.Publisher('/mobile_base_controller/cmd_vel', Twist, queue_size=10)
34 # create the publisher, tell it which topic to publish on
35 arm_command_publisher = rospy.Publisher('/arm/joint_position_controller/command', Float64MultiArray,
       queue_size=10)
36 #create the service, serviceproxy is aan de client side, dus voor het verzenden
37 set_gripper_angle_service = rospy.ServiceProxy('/mirte/set_servoGripper_servo_angle', SetServoAngle,
       persistent = True)
38 set_pincher_servo_angle = rospy.ServiceProxy('/mirte_remote/mirte/set_gripper_cntlr_servo_angle',
       SetServoAngle, persistent = True)
39
40
41 # ---- General values ----------
42 update_speed = 0.1
43 stiffness_gripper_spring = 2.08
44 yaw_offset = 0.51 #yaw on magnet
45 shoulder_offset = 2.1 #straight up
46 elbow_offset = 3.13 - 0.5 #straight up
47 wrist_offset = 4.50 #fully bent inwards
48 gripper_offset = 2.20 #fully open
49 servo_offset = math.radians(130) #graden
50 spring_engagement_angle = 0.60
51
52
53 # ---- Get message data from subscription
54 def callback_states(msg):
55     global joint_states_data
56     joint_states_data = msg.position
57
58 def get_switch_values():
59     return switch_1.value.data, switch_2.value.data, switch_3.value.data
60
61
62 # ---- Define sensors on controller ----------
63 yaw_encoder = YawEncoder('/mirte_remote/mirte/encoder/encoders/a', spring_engagement_angle,
64                          calibrating_angle = 0, scaling_factor = 1, first_order_control_gain = 2, offset
       = yaw_offset)
65 schoulder_encoder = Encoder('/mirte_remote/mirte/encoder/encoders/b',
66                          calibrating_angle = 0, offset = shoulder_offset)
67 elbow_encoder = Encoder('/mirte_remote/mirte/encoder/encoders/c', scaling_factor = -1,
68                          calibrating_angle = 0, offset = elbow_offset)
69 wrist_encoder = Encoder('/mirte_remote/mirte/encoder/encoders/d', scaling_factor = -1,
70                          calibrating_angle = 1.28, offset = wrist_offset)
71 gripper_encoder = GripperEncoder('/mirte_remote/mirte/encoder/encoders/e',
72                          calibrating_angle = 0, offset = gripper_offset, scaling_factor = 1) #checken
73 joystick = Joystick('/mirte_remote/mirte/intensity/x_axis',
74                          '/mirte_remote/mirte/intensity/y_axis', 0.5 * 1/2040)
75 gripper_simulator = GripperSimulator()
76 #joint_states = rospy.Subscriber('/arm/joint_states', JointState, callback_states)
77
78 switch_1 = Switch("20")
79 switch_2 = Switch("19")
80 switch_3 = Switch("18")
81
82 # ---- Define sensors on Mirte Master
83 mirte_yaw_joint_state = Joint('/mirte/servos/servoRot/position')
84 mirte_schoulder_joint_state = Joint('/mirte/servos/servoShoulder/position')
85 mirte_elbow_joint_state = Joint('/mirte/servos/servoElbow/position')
86 mirte_wrist_joint_state = Joint('/mirte/servos/servoWrist/position')
87 mirte_gripper_joint_state = Joint('/mirte/servos/servoGripper/position')
88
89
90 # ---- define robot zero states ----------
91 base_zero = Twist()
92 base_zero.linear.x = 0.0
93 base_zero.linear.y = 0.0
```

```
base_zero.angular.z = 0.0

arm_up = Float64MultiArray()
arm_up.data = [0,  0.9547413253784178, -2.0481974124908446, -1.418008804321289]

gripper_open = 0.5
pincher_free = 0
# set_pincher_servo_angle(pincher_free)

# ----- Set everything to start position --------
set_gripper_angle_service(gripper_open) #-0.3 is een vinger ertussen
arm_command_publisher.publish(arm_up)
rospy.sleep(1)
rospy.loginfo('Robot is ready to go!')

print(f"switch_1.value {switch_1.value.data}, switch_2.value {switch_2.value.data}, switch_3.value {switch_3.value.data},")
rospy.sleep(0.1)

# ----- Calibrate joystick
# print("press enter to calibrate yaw on magnet:")
# ready = input()
# yaw_encoder.calibrate()

# print("press enter to calibrate shoulder straight up:")
# ready = input()
# schoulder_encoder.calibrate()

# print("press enter to calibrate elbow straigt up:")
# ready = input()
# elbow_encoder.calibrate()

# print("press enter to calibrate wrist fully bent inwards:")
# ready = input()
# wrist_encoder.calibrate()

# print("press enter to calibrate gripper fully open:")
# ready = input()
# gripper_encoder.calibrate()

rospy.sleep(0.1)
print('Controller is calibrated! Wil start controlling now!')

def main_loop():
    # ----- Data types for actuation data --------
    arm_state = Float64MultiArray()
    base_speed = Twist()
    time_last_switch_check = time.time() - 0.5

    i = 0
    while i < 4000:
        i += 1

        if time.time() - time_last_switch_check > 0.5:
            val_switch_1, val_switch_2, val_switch_3 = get_switch_values()
            time_last_switch_check = time.time()

        #----- get base input --------
        if val_switch_1 == 1:
            base_speed.linear.x = (-joystick.x_value)
            base_speed.linear.y = (joystick.y_value)
            base_speed.angular.z = 0.0 #for holonomic, change into if statement for other switch value
        else:
            base_speed.linear.x = (-joystick.x_value)
            base_speed.linear.y = 0.0
            base_speed.angular.z = 3*(joystick.y_value)

        #----- get arm input --------
        if val_switch_2 == 1:
            arm_state.data = [yaw_encoder.value, -schoulder_encoder.value, -elbow_encoder.value, -wrist_encoder.value]
        else:
            arm_state.data = [yaw_encoder.value, schoulder_encoder.value,
                      elbow_encoder.value, wrist_encoder.value]

        #----- get gripper input --------
        pincher_angle = gripper_encoder.value
        actuation_mirte_gripper = gripper_encoder.actuation_mirte_gripper

        #----- get mirte gripping force--------
        gripper_simulator.set_gripper_angle(actuation_mirte_gripper)
        gripping_force = gripper_simulator.get_gripping_force()
        desired_spring_displacement = gripping_force / 16* stiffness_gripper_spring
        pincher_servo_angle = math.degrees(-pincher_angle + desired_spring_displacement+servo_offset)

        #add publish command for controller gripper servo
        base_command_publisher.publish(base_speed)
        arm_command_publisher.publish(arm_state)
        set_gripper_angle_service(actuation_mirte_gripper)

        if pincher_servo_angle < 0:
            pincher_servo_angle = 0
        set_pincher_servo_angle(pincher_servo_angle)

# ----- calling the control function --------
main_loop()


# ----- Collect analyzing data --------
test_data = {"t_yaw_enc": yaw_encoder.timestamps, "data_yaw_enc": yaw_encoder.data,
                    "t_yaw_mirte": mirte_yaw_joint_state.timestamps, "data_yaw_mirte": mirte_yaw_joint_state.data,
                    "t_schoulder_enc": schoulder_encoder.timestamps, "data_schoulder_enc":
```

```
    schoulder_encoder.data,
194                 "t_schoulder_mirte": mirte_schoulder_joint_state.timestamps, "data_schoulder_mirte":
     mirte_schoulder_joint_state.data,
195                 "t_elbow_enc": elbow_encoder.timestamps, "data_elbow_enc": elbow_encoder.data,
196                 "t_elbow_mirte": mirte_elbow_joint_state.timestamps, "data_elbow_mirte":
    mirte_elbow_joint_state.data,
197                 "t_wrist_enc": wrist_encoder.timestamps, "data_wrist_enc": wrist_encoder.data,
198                 "t_wrist_mirte": mirte_wrist_joint_state.timestamps, "data_wrist_mirte":
    mirte_wrist_joint_state.data,
199                 "t_gripper_enc": gripper_encoder.timestamps, "data_gripper_enc": gripper_encoder.
    data,
200                 "t_gripper_mirte": mirte_gripper_joint_state.timestamps, "data_gripper_mirte":
    mirte_gripper_joint_state.data}
201 df = pd.DataFrame(dict([(key, pd.Series(value)) for key, value in test_data.items()]))
202
203 df.to_csv('joint_data.csv', index=False)
```

Listing 1: Main controller script

Appendix E Classes code

```
import rospy
import numpy as np
import time
from datetime import datetime

# load the specific message format required for joint commands
from std_msgs.msg import Float32
from mirte_msgs.srv import GetPinValue
from mirte_msgs.msg import Intensity, ServoPosition

t0 = time.time()

class Encoder():

    def __init__(self, path, scaling_factor=1, calibrating_angle=None, offset = 0):
        self.subscriber = rospy.Subscriber(path, Float32, self.msg_reciever)
        self._value = None
        self.raw_value = None
        self.calibrating_angle = calibrating_angle
        self.scaling_factor = scaling_factor
        self.offset = offset
        self.raw_data = []
        self.data = []
        self.timestamps = []

    def msg_reciever(self, msg):
        raw_value = msg.data
        timestamp = time.time() - t0

        if self.offset is None:
            return
        value = (raw_value - self.offset)
        while value > np.pi:
            value -= 2 * np.pi
        while value < -np.pi:
            value += 2 * np.pi
        value = value * self.scaling_factor
        self.value = value
        self.raw_value = raw_value

        self.raw_data.append(raw_value)
        self.data.append(value)
        self.timestamps.append(timestamp)


    # def calibrate(self):
    #     if self.calibrating_angle is None:
    #         raise ValueError("there is no calibrating angle defined for this encoder")
    #     if self.raw_value is None:
    #         raise ValueError("no encoder data available to calibrate on")
    #     self.offset = self.raw_value - self.calibrating_angle
    #     print('off', self.offset, 'raw', self.raw_value, 'cali', self.calibrating_angle)


    def get_value(self):
        return self._value

    def set_value(self, val):
        self._value = val


    value = property(fget = get_value, fset = set_value)

class GripperEncoder(Encoder):
    def __init__(self, path, scaling_factor=1, calibrating_angle=None, offset = 0):
        super().__init__(path, scaling_factor, calibrating_angle, offset)

    def get_actuation_mirte_gripper(self):
        pincher_range = 2.27
        if not self._value:
            return 0
        return (-self._value * (1/pincher_range) + 0.5)

    actuation_mirte_gripper = property(fget=get_actuation_mirte_gripper)

class YawEncoder(Encoder):
    def __init__(self, path, spring_engagement_angle, scaling_factor=1, calibrating_angle=None,
     first_order_control_gain = 1, offset = 0):
        super().__init__(path, scaling_factor, calibrating_angle, offset)
        self.scroll_offset = 0
        self.first_order_control_gain = first_order_control_gain
        self.getvalue_timestamp = None
        self.spring_engagement_angle = spring_engagement_angle * scaling_factor

    def set_yaw_value(self, val):
        self._value = val

        if self.offset is None:
            return

        if self.getvalue_timestamp is None:
            dt = 0
        else:
            # if datetime.now().second - self.getvalue_timestamp.second > 1:
            #     raise RuntimeError("Too mutch time between yaw encoder value updates")
            dt = 10e-7 * (datetime.now().microsecond - self.getvalue_timestamp.microsecond)
        if dt < 0:
            dt += 1

```

```
        self.getvalue_timestamp = datetime.now()

        spring_depression = max(0, abs(self._value) - self.spring_engagement_angle)
        if self._value > 0:
            self.scroll_offset += spring_depression * self.first_order_control_gain * dt
        else:
            self.scroll_offset -= spring_depression * self.first_order_control_gain * dt

        self.scroll_offset = min(max(self.scroll_offset, -1), 1)

    def get_yaw_value(self):
        if self._value > self.spring_engagement_angle:
            return self.spring_engagement_angle + self.scroll_offset
        elif self._value < -self.spring_engagement_angle:
            return -self.spring_engagement_angle + self.scroll_offset
        # print(f"_value {self._value}, springengagementangle {self.spring_engagement_angle}, scroll offset {self.scroll_offset}, total {self._value + self.scroll_offset}")
        return self._value + self.scroll_offset

    value = property(fget = get_yaw_value, fset = set_yaw_value)

class Joystick():

    def __init__(self, path_x, path_y, scaling_factor):
        self.subscriber_x = rospy.Subscriber(path_x, Intensity, self.msg_reciever_x)
        self.subscriber_y = rospy.Subscriber(path_y, Intensity, self.msg_reciever_y)
        self.x_value = None
        self.y_value = None
        self.raw_x_value = None
        self.raw_y_value = None
        self.scaling_factor = scaling_factor
        self.x_offset = 1600
        self.y_offset = 1600

        rospy.sleep(0.4)
        self.calibrate()

    def msg_reciever_x(self, msg):
        raw_x_value = msg.value
        x_value = (raw_x_value - self.x_offset) * self.scaling_factor
        if abs(x_value) < 0.1:
            x_value = 0

        self.raw_x_value = raw_x_value
        self.x_value = x_value

    def msg_reciever_y(self, msg):
        self.raw_y_value = msg.value
        self.y_value = (self.raw_y_value - self.y_offset) * self.scaling_factor
        if abs(self.y_value) < 0.1:
            self.y_value = 0

    def calibrate(self):
        if self.x_value is None or self.y_value is None:
            raise ValueError("no encoder data available to calibrate on")
        self.x_offset = self.raw_x_value
        self.y_offset = self.raw_y_value

class Switch():

    def __init__(self, pin_number: str):
        self.service = rospy.ServiceProxy('mirte_remote/mirte/get_pin_value', GetPinValue, persistent = True)
        self.pin_number = pin_number

    value = property(fget = lambda self: self.service(self.pin_number, "digital"))

class Joint():
    def __init__(self, path):
        self.subscriber = rospy.Subscriber(path, ServoPosition, self.msg_reciever_states)
        self.data = []
        self.timestamps = []

    def msg_reciever_states(self, msg):
        self.timestamps.append(time.time() - t0)
        self.data.append(msg.angle)
```

Listing 2: Classes script, defining the different transformations for mobility

Appendix F Simulation code

```python
from math import pi
import time

object_width = 30 #mm
object_spring_constant = 2 #N/mm
max_force = 10 #Newton
scaling_factor = 3

class GripperSimulator:
    def __init__(self): # ----- Initial gripper parameters ---------
        self.gripper_distance = None
        self.commanded_opening_distance = None
        self.exerted_force = 0

    def set_gripper_angle(self, angle: float): # ----- Called function where gripper control is communicated ---------
        """Set the horizontal distance between gripper fingers.

        -angle(float): angle published to mirtes gripper -0.5 to 0.5
        """

        # ----- Calculate size of opening of Mirte Master gripper and respective force ---------
        opening_distance = (angle + 0.5) * 120 # 0 to 50 mm
        self.commanded_opening_distance = opening_distance
        self.exerted_force = round(max(0, (object_width - opening_distance) * object_spring_constant), 3)

        if self.exerted_force < max_force:
            self.exerted_force = self.exerted_force
        else:
            self.exerted_force = max_force

    # ----- Called function for force ---------
    def get_gripping_force(self):
        return self.exerted_force
```

Listing 3: Simulation code, made to simulate the force in the gripper of the Mirte Master

Appendix G Deviation graphs

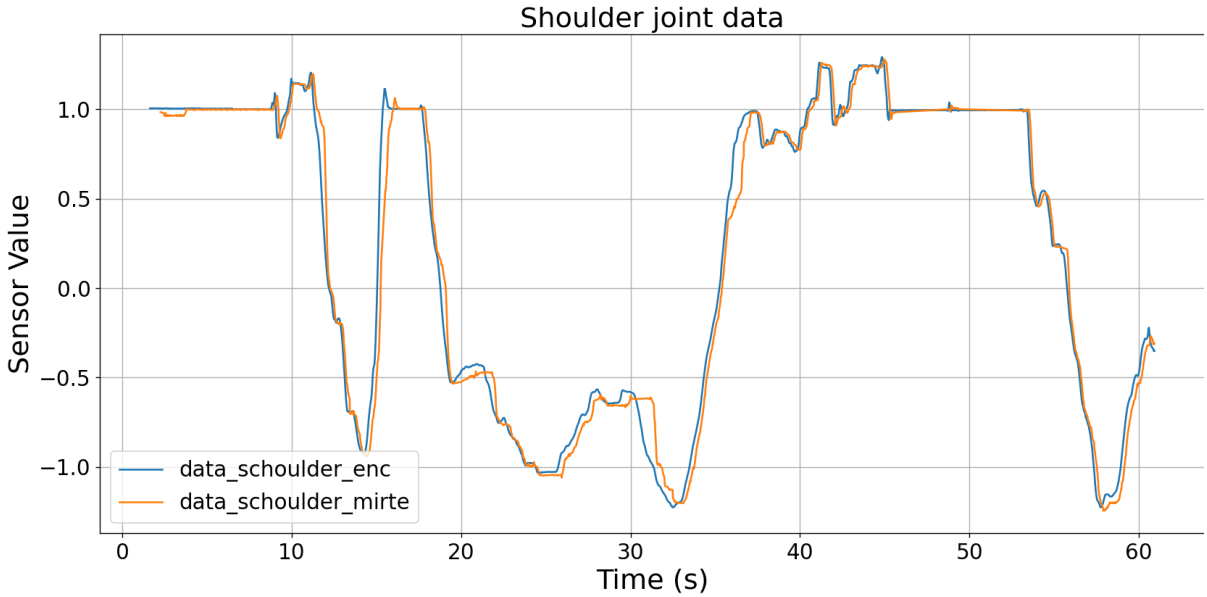


Figure 20: *The shoulder joint comparison shows that the output closely follows the input. Looking at the graph shows a maximum deviation of roughly 0.85 radians.*

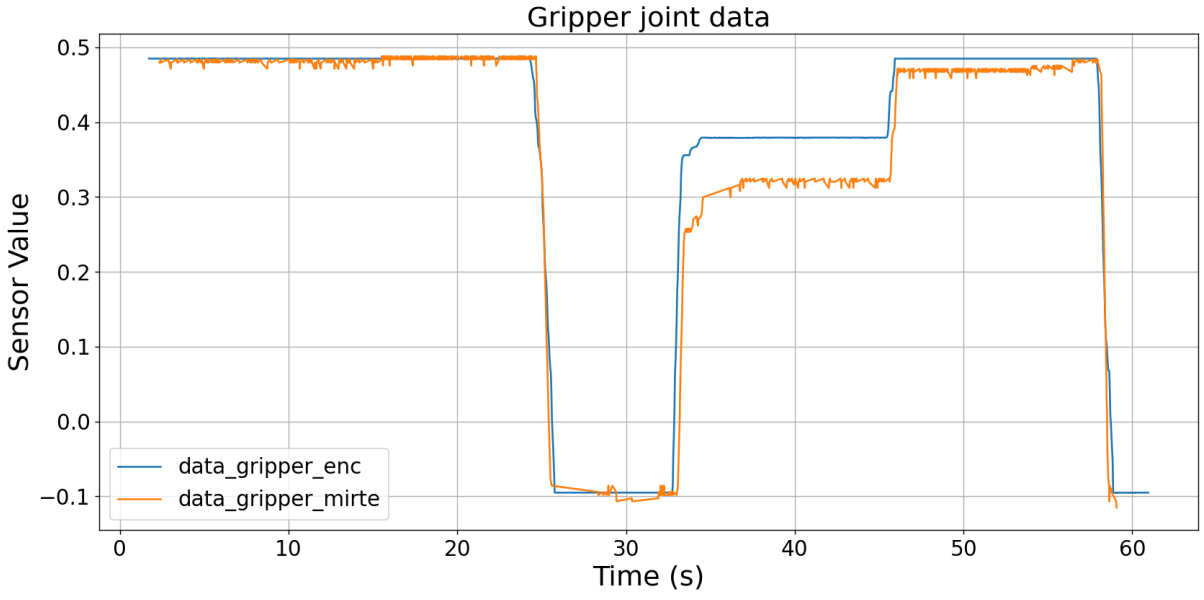


Figure 21: *When comparing the pincher and gripper data it shows about a maximum deviation of roughly 0.08 radians.*

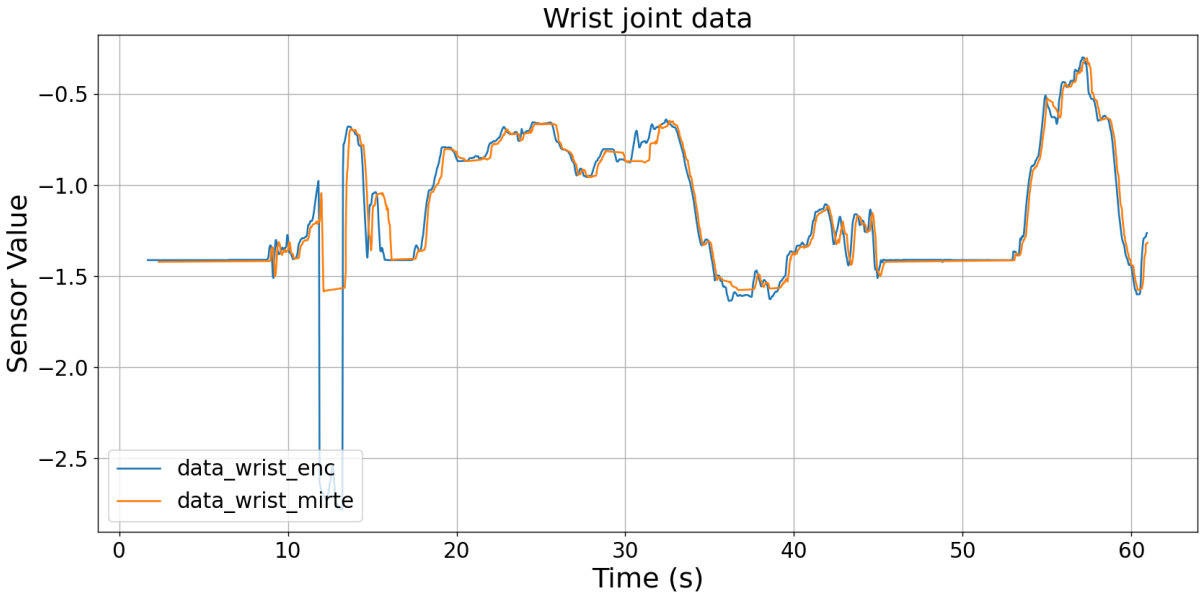


Figure 22: *The output follows the input very closely. Excluding the outlier gives for a maximum deviation of around 0.035*

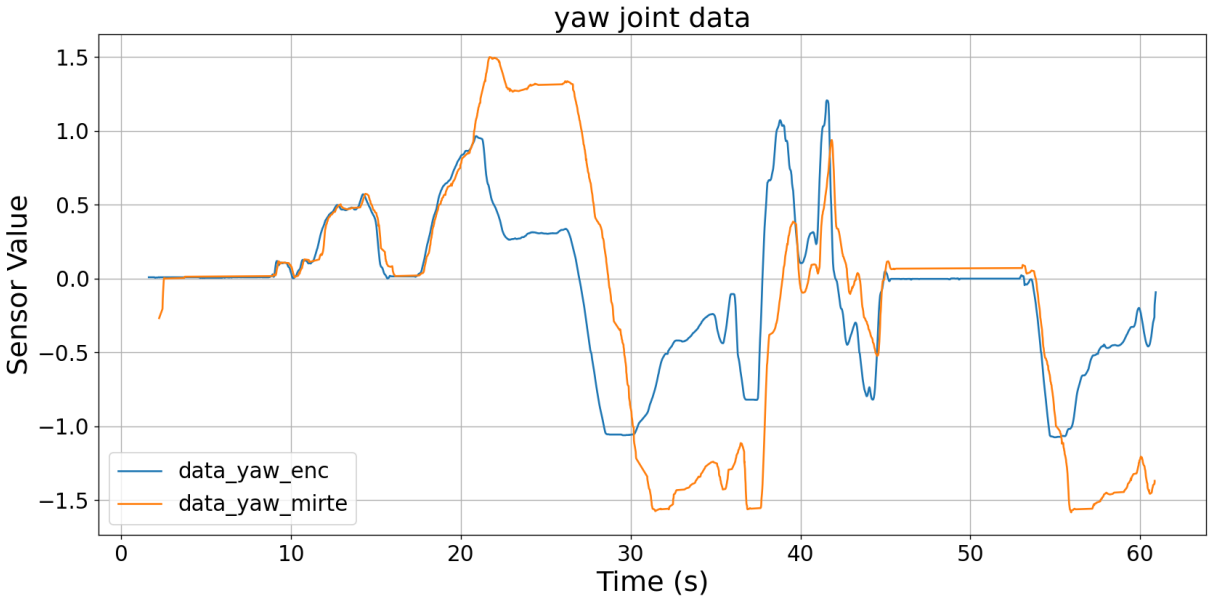


Figure 23: *Exluding the outliers because of the toggle mode, compared to the other joints, the output on average deviates more.*

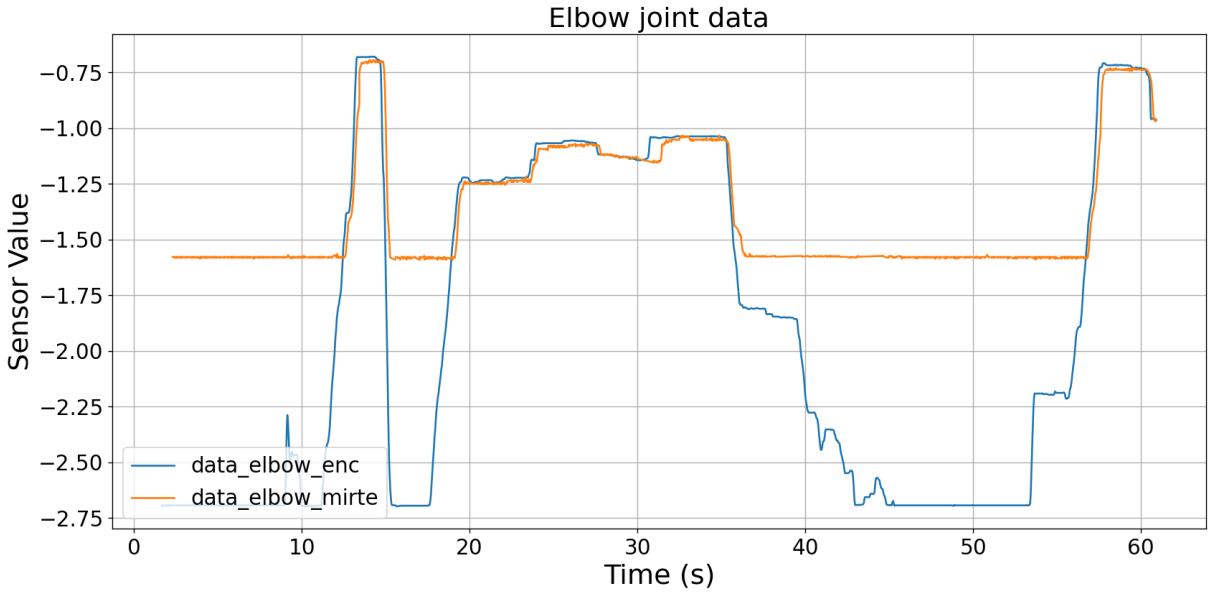


Figure 24: *Some sections clearly don't align which is because of the workspace of the MM. This is normal and doesn't cause issues when teleoperating. Ignoring these sections gives a max deviation of roughly 0.3 radians.*

Appendix H Design Process

After researching the existing teleoperation interfaces, a morphological chart was created, as seen in the appendix table 1. To narrow down all the possible design options in the morphological chart, three concept designs were chosen that aligned with the design requirements that were discussed in the introduction.
The first design is a controller where the casing is based on the N64 controller with the middle grip:

Figure 25: *Nintendo 64 (N64) controller*

It includes a small controller arm, which is a scaled-down version of the MM's arm, except without a yaw joint. It also includes a force-actuated button on the end-effector to control the gripper. The yaw is controlled by a joystick. The controller arm is a scaled-down version of the MM arm.
The second design is based on the existing design of the Force dimension delta 3 with an end-effector with a scroll wheel to control the wrist. This end effector can control the position of the gripper on the MM. The pincher will be located on the end effector.
The third design includes a controller arm, which is again a scaled-down version of the MM arm with limited yaw, where a pinching mechanism for controlling the gripper is located separately on the case. Multiple toggle switches are also present, which can have various usages.
All concepts include a joystick to control the base and are designed to comply with the design requirements. Five criteria were specified to determine which of the three concepts is best for our application. Each design was rated based on the criteria as seen in table 2. The best rated design is the (Design 2 (Q)), because it scored best on User Friendly and Reliability.

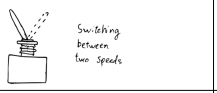
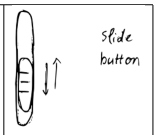
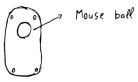
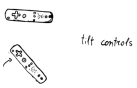
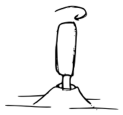
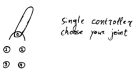
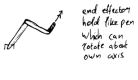
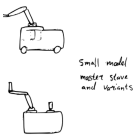
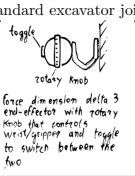
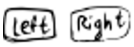
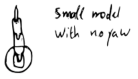
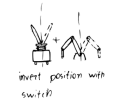
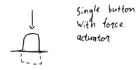
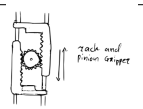
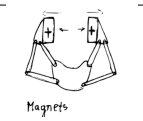
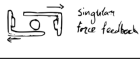
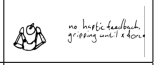
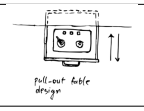
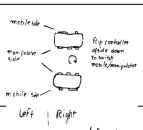
Base	Speed					
	Direction					
						
Arm Position	Combined Solutions			standard excavator joint		
						
	Shoulder Yaw Joint					
	Shoulder Pitch Joint					
	Elbow and Wrist Joint	Already covered in Combined Solutions				
Gripper	Gripper					
						
	Force Feedback					
General	Casing					
	Button layout					
	Controller mobility					
	State Observing					

Table 1: The Morphological Table, used to create multiple designs as proof of concepts

	Weights (%)	Design 1 (Y)	Design 2 (Q)	Design 3 (D)
User Friendly	30	5	8	6
Manufacturability	15	3	8	4
Sustainability	10	5	6	6
Operational Efficiency	20	8	7	7
Reliability	25	7	7	6
Total	100	5.8	7.4	5.9

Table 2: *Weighted Criteria Table. User Friendly has the heaviest score, because the operator will be carrying the teleoperation device for longer periods of time. Reliability is the second heaviest because it is important for it not to break. Looking at the total score the winner is "Design 2 (Q)".*