7MBI0101 Advanced Medical Robotics Coursework Report (2024) Haptic Device – Team 5

The Hung, Dang

Abstract—Haptics is an important sensor that increases the information received and precision of human tasks. The project aims to create a haptic interface from a three degrees-of-freedom planar robot operated by a Raspberry Pi. An environment generation program was developed to create virtual environments. Virtual environments were created in advance for the haptic interface by discretizing the workspace into 1mm2 pixels that store the environment parameter. The device would render the force at the end-effector using these parameters. Visualization was done using Remote Desktop Connection. User tests were done to assess the device's performance. In the end, we succeeded in creating a scalable and modifiable virtual environment with a friendly guided user interface as well as a haptic device working in that environment. The haptic calculation could be done with a frequency of 529Hz, while the communication of the hardware was limited to 40Hz. Real time visualization was done at 18Hz. User testing showed that the device could simulate the intended environment created by the developers.

Index Terms: environment simulation, force control, Jacobian-based control, haptics

I. INTRODUCTION

HAPTIC, which is normally conceived as comprising both kinesthetic and tactile perception, plays a crucial role as one of the most important sensors in the human body [1]. It not only helps a person get information about the world, but it also closes the control loop in many situations, like holding or grasping an object [2-4]. A haptic interface is a device that delivers the user with a haptic feeling, either from a virtual environment or relayed from a distant location [5, 6].

The importance of haptics, especially in medicine, is unneglectable. In robot-assisted surgery, lacking haptics feedback prolongs the surgery time, which exposes the patient to more risk of infection, blood loss, and many other complications [7, 8]. Although some experienced surgeons could perform surgery well without touch feeling and the evidence backing the benefit of haptics devices is not strong at the moment [9], haptic interface development, especially in master-slave surgical robots where surgeons' feedback information is even more limited, still receives lots of concerns from the scientific communities [10]. One more valuable application of the haptics device is medical education. As the human body is very fragile, many procedures are not allowed to be practiced unless the practitioners have enough training and experience; some signs and diseases might not even be common enough for healthcare workers to practice on. In these situations, haptic devices have become a very potential training tool to help healthcare practitioners get familiar with the work [11, 12]. Moreover, in patients' examinations, touching is a very important step [13, 14]. By touching, a doctor can get information about both surface and deep lesions and use it to make diagnoses. A haptic device can transmit this information from far away, facilitating the development of telemedicine [15].

Virtual fixtures are a high-level control concept. It impacts robot kinematics and dynamics by adding more rules for robot movement within its workspace [16]. Combining virtual fixtures with a haptics interface, one can simulate different real-world objects and environments, or even rules that do not exist in the physical world. In this project, we will create a virtual environment with multiple virtual fixtures at different locations. Using a three-degree-of-freedom (3-DOF) planar robot, we will create a haptic device by evaluating these virtual fixtures at each iteration and generating the force through a force controller. Additionally, before building the haptic device, we will build a task space controller for the robot, which will be helpful for validation tasks. The projects could be decomposed into five main components: Jacobian-based task-space controllers with linear pathway generation for 3-DOF planar robots, a force controller that is properly tuned with reference to real-world force value, a haptic device with a scalable haptic rendering mechanism that can simulate multiple environments in real time in one run, a user-friendly environment generation program that can be used to create and modify the virtual environment, and a and a real-time visualization method for the haptic device and the environment.

As our robot is a 3-DOF robot, there will be many solutions for one end-effector configuration. Jacobian-based controllers transform the difference in the task space into joint space while minimizing the norm of the joint velocity vector [17]. The force controller in our project referred to the direct control of the effector force through controlling the joint torque [18]. The friction and dynamics of the robot were identified to increase the precision of the controller. To store the virtual fixtures, the workspace of the robot was discretized into $1mm^2$ pixels, where

each pixel stores different parameters of the environment. This will offer flexibility in the types and number of virtual fixtures that can be stored. Haptic rendering refers to the process of calculating the force and exerting it on the user. Utilizing the additivity of force, the controller will render the haptics by summing all the effects of the virtual fixtures for the user.

II. Methodology

A. *Equipment, and setup*

The project hardware included three Dynamixel XC330-M288-T motors integrated with three 3D-printed robot arms and one base. Forward kinematics is given in Appendix 1. The robot workspace is a semicircle with a radius of 300mm, bounded by a 300x600mm rectangular (Fig. 1).

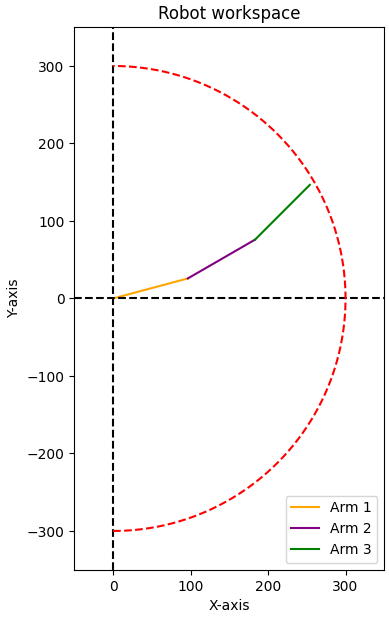

Fig. 1. Robot workspace is a semicircle with a radius of 300mm.

Three motors were connected to an OpenRB-150 embedded controller. The microcontroller was connected to a Raspberry Pi 4 running Linux to control three motors. The controllers were coded in the ROS 2 environment on the Raspberry Pi. The motors will send the measured angles and current to the microcontroller, which are relayed to the Raspberry Pi. The Raspberry Pi will send the desired electrical current value to the microcontroller, and the microcontroller will control motors with this value. The communication between the Raspberry Pi and the microcontroller was limited to 40Hz. The current values sent to the motors were limited to being below 500mA. The relationship between torque and current are handled by Dynamixel internal controller and assumed to have a linear relationship (1).

$$\tau_{\text{act}} = K_i \cdot i \tag{1}$$

Where τ_{act} is the torque (N.m) of the motor, i is the current (mA) sent to the motor, and K_i is a constant representing the linear relationship. Data are collected, validated, and presented as mean ± standard deviation (SD), coefficient of determination R^2, or any metrics mentioned in that specific section. A summary of the hardware can be found in Table 1.

TABLE I
COMPONENTS USED FOR THE ROBOT

Components	Number
Raspberry Pi 4	1
OpenRB-150	1
Dynamixel XC330-M288-T motors	3
3D printed robot arms and base	3
3D printed robot base	1
Wires and cable connectors	

B. *Friction modelling*

Simple PI controllers were used to control each robot motor velocity separately (Fig. 2).

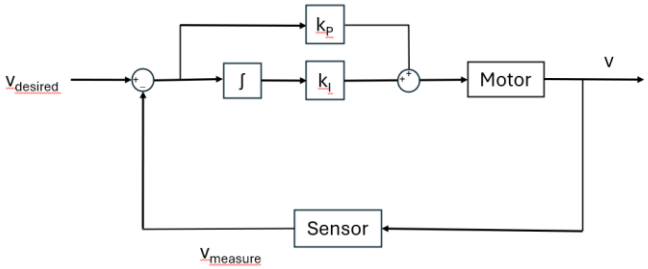

Fig. 2. Initial PI controller.

Using the PI controller, each motor was instructed to move at different constant velocities. The values of current were plotted against velocity. The friction model is assumed to be only dependent on velocity and estimated using equation (2).

$$\tau_{\text{friction}} = K_i \cdot \text{i}_{\text{friction}}$$
$$= K_i \cdot \left(coeff_1 \cdot \dot{q} + coeff_2 \cdot sign(\dot{q})\right) \tag{2}$$

Where τ_{friction}, $i_{friction}$ are the torques (N.m) and current (mA) exerted by the motors to counter the friction torques, $\dot{q}$ is the velocity of the motor (degree/second), coeff_1 and coeff_1 are the coefficients of the models. The coeff_1 and coeff_2 are calculated by minimizing the sum of square errors between the measured current and the predicted current. The friction model was tested by instructing the robot to move at random speeds.

C. *Dynamic modelling*

The inverse dynamic of the robot was expressed as (3).

$$\tau_{dynamic} = B(q) \cdot \ddot{q} + c(q, \dot{q}) \tag{3}$$

Where $\text{q} = [q_1, q_2, q_3]$, $\dot{\text{q}} = [\dot{q_1}, \dot{q_2}, \dot{q_3}]$, $\ddot{\text{q}} = [\ddot{q_1}, \dot{q_2}, \ddot{q_3}]$ represent the joint angles, joint velocities, and joint accelerations. $\text{B}(\text{q})$ is the inertia matrix, and $\text{c}(\text{q}, \dot{\text{q}})$ is the Coriolis terms at joint states $\text{q}, \dot{\text{q}}, \ddot{\text{q}}$. $\tau_{dynamic} = [\tau_{dynamic_1}, \tau_{dynamic_2}, \tau_{dynamic_3}]$ are the torque exerted by the motors to have the corresponding dynamic behaviors. The $\text{B}(\text{q})$, $\text{c}(\text{q}, \dot{\text{q}})$ model was calculated using Lagrangian method. The potential energy was considered to be zero, and the kinetic

energy was parametrized using joints angles q, arm lengths $l = [l_1, l_2, l_3]$, distances from arm origins to center of mass $l_c = [l_{c1}, l_{c2}, l_{c3}]$, arm weights $m = [m_1, m_2, m_3]$, and arm moment of inertia $I = [I_1, I_2, I_3]$ (Fig. 3). To calculate the position of the center of mass and moment of inertia of each arm, the robot links were assumed to be rectangular prisms, while the motors were assumed to be cylinders.

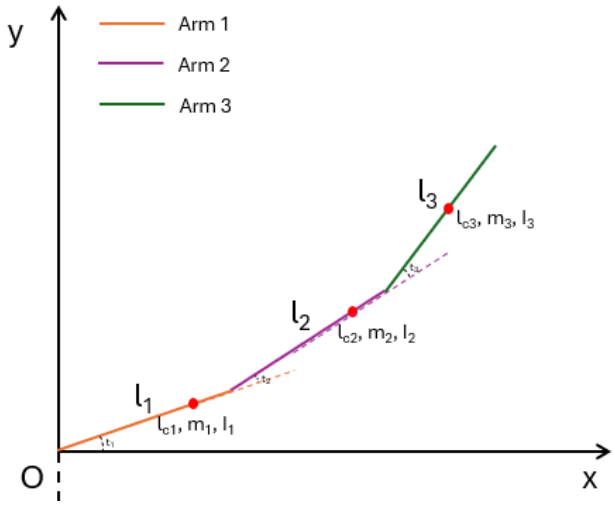

Fig. 3. Parameters for calculating robot dynamic model.

The result model was mathematically validated by inputting different joint states $q, \dot{q}, \ddot{q}$, and comparing the output $\tau_{dynamic}$ with expected $\tau_{dynamic}$. Additionally, the robot was instructed to move between two configurations, and the model was validated by how well it could predict the current exerted by the motors.

D. *Velocity PID controller*

Using the friction model, a PID controller with a feedforward path was used for the final velocity controller for each motor (Fig. 4). The steady-state error of each motor were used to assess its performance.

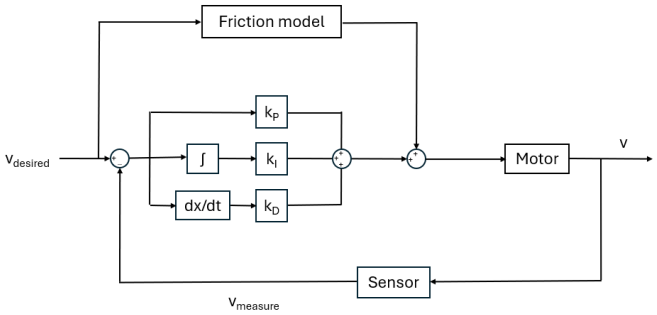

Fig. 4. Final controller for joint velocities.

E. *Task space controller*

Five ROS nodes were used to run the controller (Fig. 5).

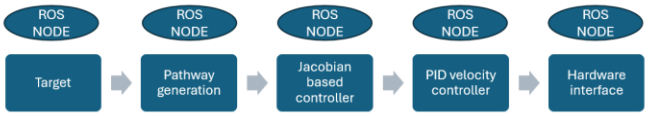

Fig. 5. ROS architecture for task space controller.

The "Hardware interface" node worked as an interface with the microcontroller and motors to collect current joint angles, joint velocity, joint current, and control the output current to the motors. Target is input from the user in the "Target" node; the node then publishes the target to the "Pathway generation" node. "Pathway generation" used the target and the current position (subscribed from "Hardware interface") to generate a set of via points along a linear path. Each via point is sent to the "Jacobian based controller" node, where the difference in task space position (dx) is used to calculate the joint velocities ($\dot{q}$) needed to move to each via point. Formula for the mapping from dx to $\dot{q}$ is given in (4).

$$\dot{q} = J^*(x) = W^{-1}J^T(JW^{-1}J^T + k^2I)^{-1} \cdot \text{clamp}(dx) \quad (4)$$

$$\text{clamp}(dx) = \begin{cases} \text{dx}, & \text{if } ||\text{dx}|| < d_{max} \\ \frac{\text{dx}}{||\text{dx}||} d_{max}, & if\ ||\text{dx}|| \geq d_{max} \end{cases} \quad (5)$$

Where J is the Jacobian matrix (Appendix 2), W is a diagonal matrix representing the weight of each joint, k is a damping factor to avoid singularity, $\text{clamp}(dx)$ clamps the $||\text{dx}||$ below predefined d_{max} (5).

Then $\dot{q}$ are sent to the "PID velocity controller" node, which is used to control the motors' velocity. Additionally, to increase precision, a PID controller is added after dx calculation. Integral clamping was used as an anti-windup technique [19]. The whole control structure is given in Fig. 6. Error distance in task space was used to assess the controller's performance.

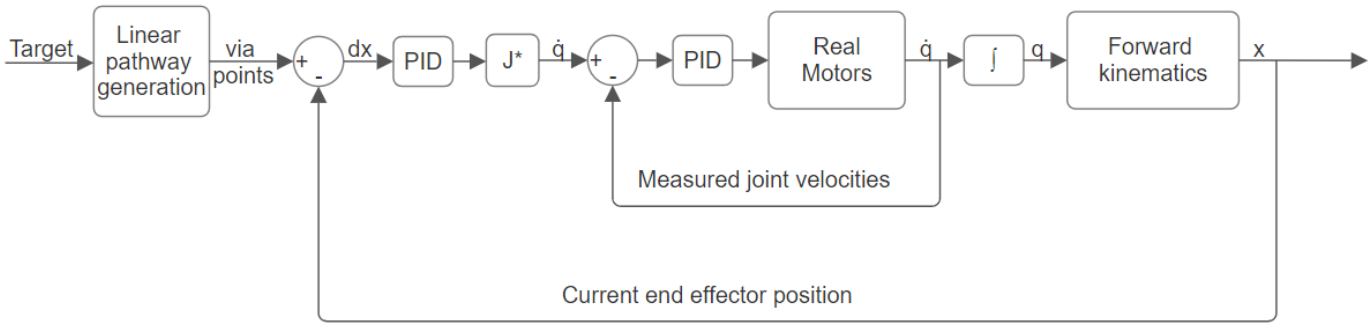

Fig. 6. Jacobian based cascade PID controller with linear pathway generation.

F. Force controller

A force controller was built using formula (6).

$$\tau_{simulated} = K_i \cdot \mathrm{i}_{simulated} = J^T \cdot f_{simulated} \tag{6}$$

Where $\tau_{simulated}$, $\mathrm{i}_{simulated}$ are the required torque and current needed to have an $f_{simulated}$ at the end effector. To find the value of K_i, arbitrary forces $\left|\frac{f_{simulated}}{K_i}\right| \in [1, 2]$ were used to calculate $\mathrm{i}_{simulated}$ at three areas in the workspace (upper, middle, lower). A load cell calibrated using known weight under gravity was used to measure the $|f_{simulated}|$. K_i was estimated using the average of the quotient of measured force and arbitrary force. To validate the accuracy of the force controller, the robot was instructed to exert different desired $f_{simulated}$ at random positions in the workspace. The desired forces were then compared to the measured forces on the load cell. The upper limit of the $f_{simulated}$ was experimentally approximated by substituting increasing $f_{simulated}$ values to the formula when the three robot arms are fully extended (three angles are all zeros) until $|\mathrm{i}_{simulated}| > 500\mathrm{mA}$.

G. Haptic device: environment simulation, haptics rendering, and visualization

To facilitate real-time implementation, a separate environment generation program was developed to create an environment to upload to ROS workspace for haptic rendering. As the robot workspace was a plane limited within a bounding rectangle of 300x600mm, the virtual environment was considered to be two-dimensional, and the environment information was stored as a three-dimensional tensor $R^{300x600xd}$, with the depth d storing different environment virtual fixture parameters. As the time complexity for accessing a tensor stored as a Python list [20] and a NumPy array [21] are both O(1), a small experiment were done to compare their access time within Raspberry Pi.

Six different types of environments were created in our environment generation program, including free space, springs, damping areas, facilitating areas, rigid walls, and rough walls. Each environment has its own parameters and is encoded differently in the depth dimension d of the environment tensor.

Free space: In free space, the motors are instructed to compensate for friction and dynamics using the aforementioned models to simulate a system with no friction and no inertia. No environment parameters were stored for free space.

Springs: given the stiffness matrix K, and spring displacement ds, $f_{simulated}$ at each position are calculated using the hook law (7). The tensor stored three parameters of this environment, including the environment type and the x and y directions of $f_{simulated}$.

$$f_{simulated} = K \cdot ds = \begin{bmatrix} K_x & K_{xy} \\ K_{xy} & K_y \end{bmatrix} \cdot ds \tag{7}$$

Smooth walls and rough walls: the walls are modeled as springs with a very high stiffness. For the rough walls, when moving along the edge, random stiffness K and random thickness were used to simulate tactile perception. Three parameters similar to the springs were stored for this environment.

Damping areas: $f_{simulated}$ are negatively proportional to the current velocity $\dot{x}$ with a factor $k_{damping}$ (8). When $k_{damping} > 0$, the areas hinder robot's movement. When $k_{damping} < 0$, the areas speed up the robot. Two environment parameters were stored involving the environment type and the $k_{damping}$ coefficient.

$$f_{simulated} = -k_{damping} \cdot \dot{x} \tag{8}$$

Facilitate areas: within this area, the robot's movement is facilitated in a predefined direction d and hindered if moved in the other direction (9). Three environment parameters – environment type and the x and y directions of d – were stored.

$$\mathrm{f}_{\mathrm{simulated}} = ||\mathrm{d} \cdot \dot{\mathrm{x}}|| \cdot \frac{\mathrm{d}}{||\mathrm{d}||} \tag{9}$$

In ROS, at each iteration, the code checks the current position of the end effector and extracts a vector of length d. The total $\mathrm{f}_{\mathrm{simulated}}$ was obtained by summing the individual forces $\mathrm{f}^i_{\mathrm{simulated}}$ from each environment i (10).

$$\mathrm{f}_{\mathrm{simulated}} = \sum_i \mathrm{f}^i_{\mathrm{simulated}} \tag{10}$$

The $\mathrm{f}_{\mathrm{simulated}}$ was transformed to $\mathrm{i}_{simulated}$ (6) and sent to the "Hardware interface" node to control the motor. The position of the robot arms and the environment were visualized using Matplotlib [22] package on a Remote Desktop Connection. The whole structure of the haptic system is given in Fig. 7.

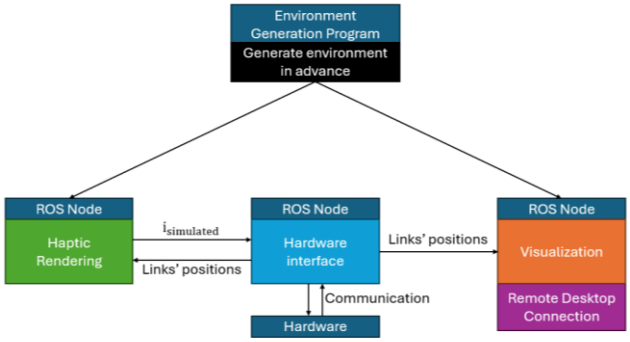

Fig. 7. Architecture of whole haptic system.

H. User testing

The performance of the haptic device was tested on humans. Three people were asked to use the GUI to create an environment with no instructions on how to use it. Six people were asked to try the device and describe the types of environments they could feel. Additionally, to evaluate the feeling without visualization, an environment with four springs with different stiffness was created. Three participants were blinded and asked whether they could guess the number of springs. After that, they are shown the visualization and asked to order the strength of the springs.

III. Results

A. Friction modelling

The friction was modeled and tested for each motor separately (Fig. 8).

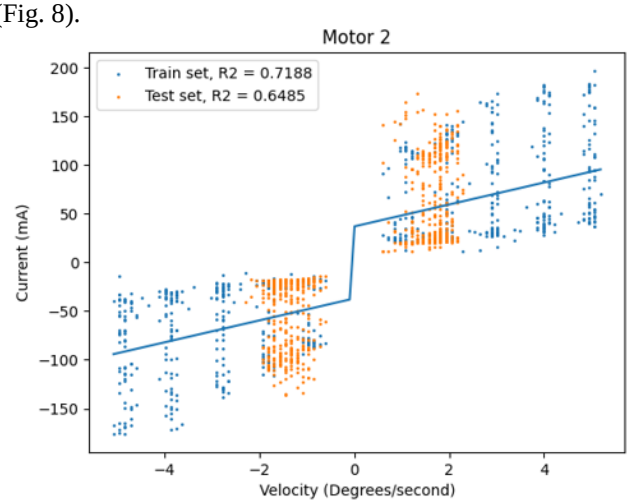

Fig. 8. Friction modelling for motor 2.

Motor 2 had significantly higher friction and variations, making the model not fit well (R^2=0.719) in comparison with motor 1 and 3 (R^2 equal 0.981 and 0.997). When tested with random movement, R^2 of the three motors reduced to 0.908, 0.649, and 0.979. A summary of the coefficients and model performance is given in Table II.

TABLE II
RESULTS OF FITTING AND TESTING FRICTION MODEL

	Motor 1	**Motor 2**	**Motor 3**
$\mathbf{coeff_1}$	2.7385	11.2785	1.0031
$\mathbf{coeff_2}$	14.9386	36.9792	15.5751
$\boldsymbol{R^2}$ **train**	0.9806	0.7188	0.9965
$\boldsymbol{R^2}$ **test**	0.9078	0.6485	0.9792

B. Dynamic modelling

The mathematical formula derived from the Euler-Lagrange equation was validated mathematically and experimentally. On mathematical validation, the model output matched the sign of expected torque (Table III).

TABLE III
MATHEMATICAL VALIDATION OF INVERSE DYNAMIC MODEL

	$\boldsymbol{q}$	$\dot{\boldsymbol{q}}$	$\ddot{\boldsymbol{q}}$	$\boldsymbol{\tau_{dynamic}}$	***Expect***
1	-60, 60, 30	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
2	60, 0, 0	10, 5, -10	0,0,0	0, 0, 0	0, 0, 0
3	0, 0, 90	0, 0, 10	0,0,0	-8.67, -4.34, 0	–, –, 0
4	0, 90, 0	0, 10, 10	0,0,0	-35.55, 0, 0	–, 0, 0
5	0, 0, 0	0, 0, 0	0, 0, 1	6.7, 4.2, 1.7	+, +, +
6	0, 0, 0	0, 0, 0	0, -1, 0	-29.3, -16.4, -4.2	–, –, –
7	0, 0, 0	0, 0, 0	1, 0, 0	59.8, 29.3, 6.7	+, +, +

**Units of $q, \dot{q}, \ddot{q}$ are degree, degree/s, and degree/s^2. Unit of $\tau_{dynamic}$ is N.m x 10^{-6}. "Expect" is the signs of expected torque are expressed as positive (+), negative (-) and zero (0).*

In the table, when the robot remains stationary (line 1), no torque is required. When the arms have to move at constant velocities with no joint accelerations (lines 2, 3, and 4), the signs of $\tau_{dynamic}$ match the signs of the torques needed to counter the centrifugal force to maintain constant velocities. Similarly, the signs still match for the torques required to accelerate each joint from its initial position (lines 5, 6, and 7). Furthermore, as the rotating axis gets closer to the origin, there is an increase in the magnitude of the required torques. These observations indicate the correctness of the dynamic model.

On real robot validation, the friction model could predict 44.0% of the robot currents, while adding a dynamic model increases the explainability by 8.2% (Fig. 9).

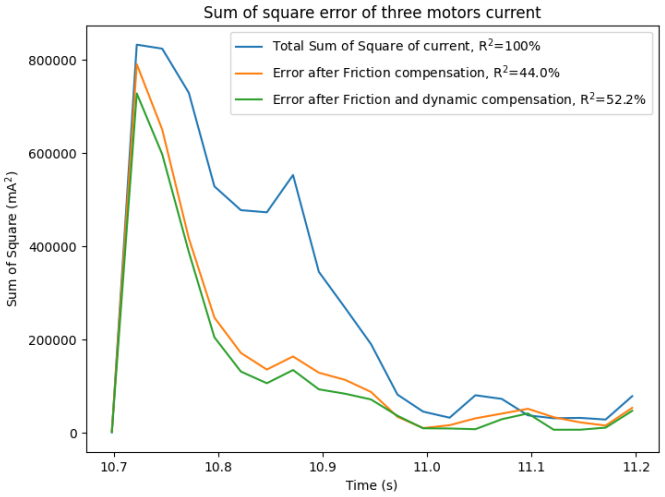

Fig. 9. Sum of square of motor currents (mA^2) of three motors from measured currents and currents prediction by friction model ± dynamic model. A perfect model would have zero errors and appear on the graph as a y = 0 line.

C. Velocity PID controller

Adding a feedforward term significantly reduced the settling time of the motors while maintaining low overshoot (Fig. 10).

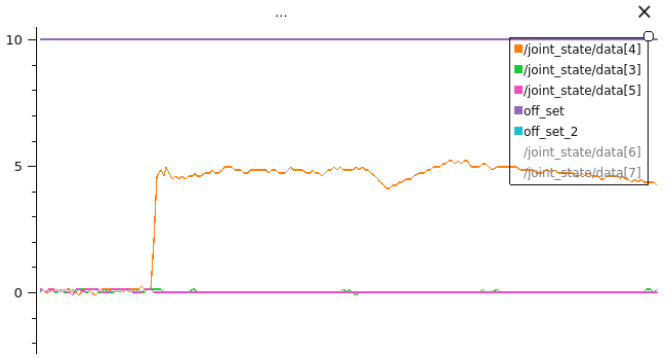

Fig. 10. Result of velocity controller with feedforward terms on the 2nd arm on the robot captured on Remote Desktop Connection, with the y axis representing the velocity and the x axis representing the time.

The (P, I, D) gain values for each motor after tuning are (1.5, 10, 0) for motors 1 and 3, and (1.5, 30, 0) for motor 2. By adding the feedforward terms, changing the P values within a wide range from 0.0 up to 5.0 did not impact the controller stability; therefore, the tuning process mainly focused on tuning the I gain values. When instructed to increase from 0 degree/s to 5 degree/s; in the steady state, the maximum errors of the PID velocity controller for the three motors are 0.5347, 0.1895, and 0.431 degree/s, respectively. The fact that motor two has a lower steady state might arise from its high friction, which accidentally acted as a physical low-pass filter for input currents.

D. Task space controller

The controller was used to move the end effector to 10 different positions. The end effector could reach a precision of 1.55mm±0.73 mm (range 0.4 – 2.9mm). To increase stability, in the final version, the controller is turned off if the error is less than 5mm (Fig. 11).

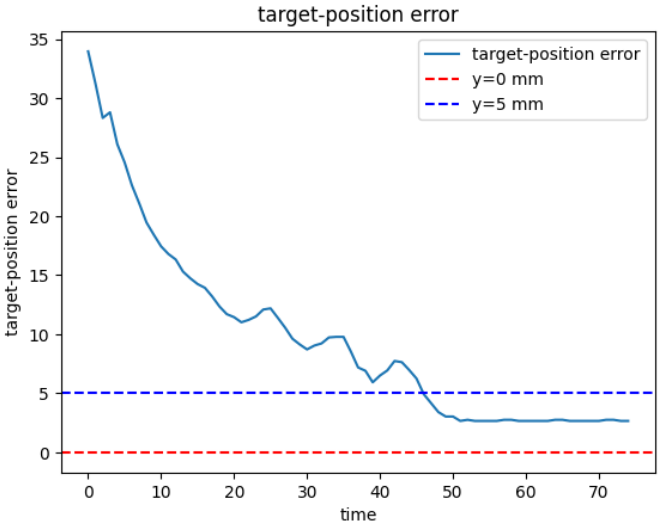

Fig. 11. Error distance from the end effector to target. The controller was turned off when the distance was less than 5mm.

E. Force controller

The load cell was working well, as during its calibration the correlation was 0.99981 (R^2 = 99.96%) (Fig. 12).

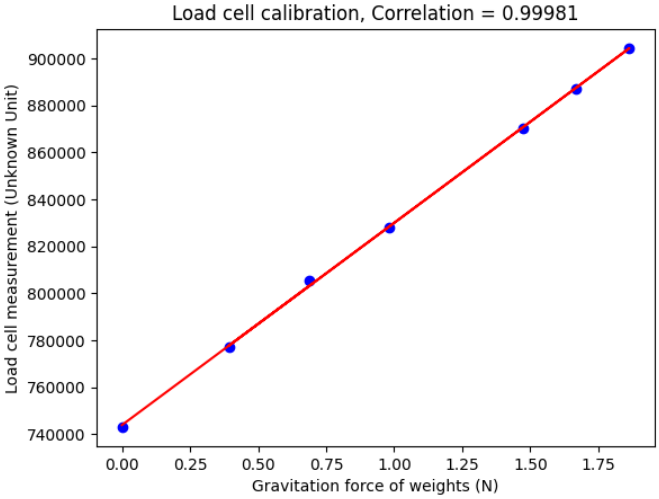

Fig. 12. Load cell calibration.

Regard the robot force calibration, the mean ± SD of K_i value was 1.487±0.228 (Fig. 13).

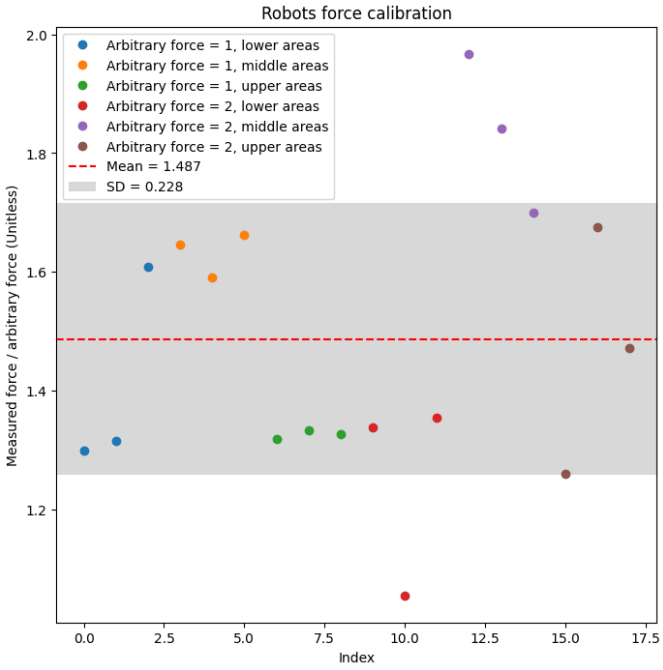

Fig. 13. Robot force controller calibration.

When the force controller was tested at random positions by exerting force in the x and y directions (Fig. 13). The mean ± SD of the error of the force controller is 0.0706 ± 0.2315 (R^2 = 83.33%), indicating a good ability to simulate real force to user.

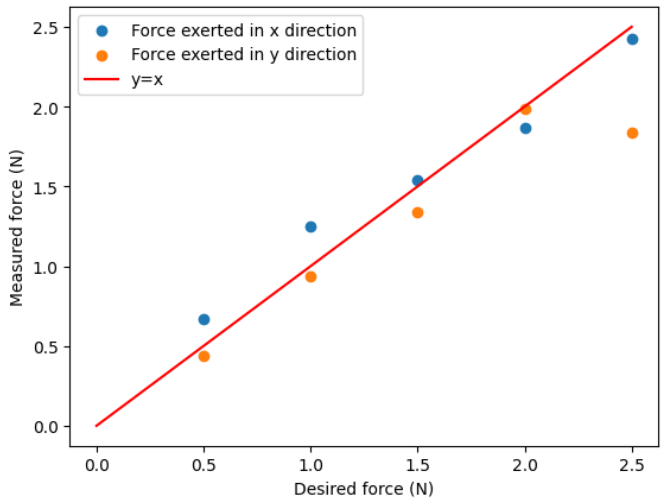

Fig. 14. Robot force controller testing.

Substituting K_i into the formula, the maximum force exerted by our robot was estimated to be 2.5N at singularity (Table IV), and therefore, all the $f_{simulated}$ from the environment simulation were attempted to be limited within this range. It is worth mentioning that the maximum $f_{simulated}$ in other configurations of the robot will be higher as the J^T will be different for each position.

TABLE IV
ESTIMATION OF MAXIMUM $f_{simulated}$

$\lvert f_{simulated} \rvert$ (N)	0	1	2	**2.5**	3
$i_{simulated}$ (mA)					
Motor 1	0	202	403	**504**	605
Motor 2	0	134	269	**336**	403
Motor 3	0	67	134	**168**	202

**All robot joints are set to be 0 (straight arms), $f_{simulated}$ is set to be orthogonal to the arms. The $\lvert i_{simulated} \rvert$ of each of three arms were limited to 500 by the hardware.*

F. Haptic device: environment simulation, haptics rendering, and visualization

On the Raspberry Pi, when accessing a 300x300x10 tensor, a Python list has a shorter time (0.847±0.039 µs) compared to a NumPy array (2.115±0.272 µs). Therefore, the environment is saved and accessed as a Python list.

The environment generation program successfully created six types of environments (including free space) (Fig. 15). The tensor used in our project to store the environment has a dimension of 300x600x6. Regarding the channel vector of length 6, one element stores the type of information, two elements store the forces at each position, one element stores the damping coefficient, and the last two store the direction vector for the facilitating areas. The environment was stackable, scalable, modifiable, and user-friendly. The program includes a guided user interface (GUI), where people select the environment (with optional parameters like spring stiffness K, damping factor $k_{damping}$, roughness of the wall, etc.) and click on the workspace (on the right) to create the environment. The "Reset" function removes all current environments; the export function exports the environment as a Python list and saves it as a pickle file. The "Validate" function prints the channel vector (vector with length d) at a selected point so the developer could check and validate the environment information at a coordinate.

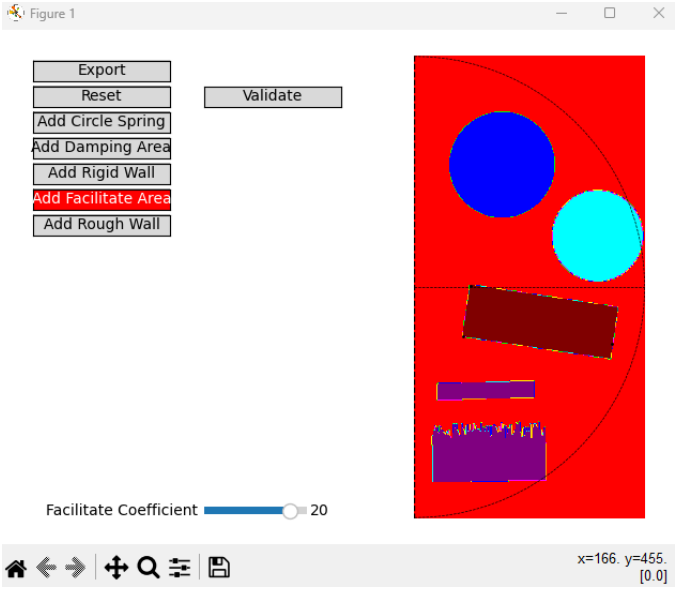

Fig. 15. GUI of environment generation program.

The haptics rendering on the Raspberry Pi took 1.918±0.071 milliseconds to calculate the $i_{simulated}$ (529 Hz). The microcontroller communicates with the hardware at a frequency of 40.006±0.001Hz; therefore, the method could perform real-time haptic rendering with the ability to extend the environment to multiple environments. Due to the additive nature of the environment, the haptics devices were still working well when stacking on each other.

To access the stability of the haptic device, we moved the end effector as fast as we could. In free space, moving the end effector up to 130mm/s (the workspace is 300x600mm), did not impact the haptic performance or stability. When other simulated obstacles are added, the researchers could move the end effectors up to 50mm/s (higher velocities were still possible, but many obstacles in the environment hindered the process). At this velocity, the haptics effect was still working well with no sign of instability.

We use the load cell to push into a spring along the y axis (Fig. 16a) and the force measured gradually increases up to 3N (Fig. 16b), showing a good transition $f_{simulated}$ generated at each pixel in the environment. The fluctuation in the measurements might be due to shaking when the researcher is holding the load cells, and the drastic changes when the force >3N might be due to $i_{simulated}$ upper limit.

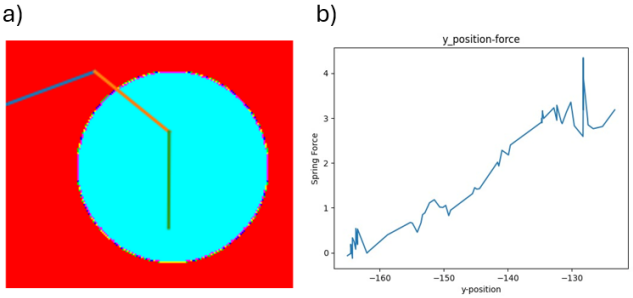

Fig. 16. The load cell was pushed into the spring in the y direction (a), and the forces measured are plotted against the y coordinates (b).

Plotting was done simultaneously on Remote Desktop Connection; the screen was updated every 0.054±0.005ms (18Hz), resulting in a smooth visualization (Fig. 17).

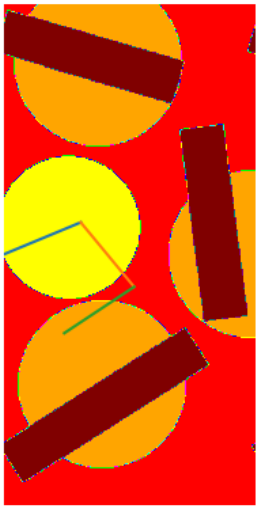

Fig. 17. Visualization on Remote Desktop Connection.

G. *User testing*

In the GUI experiment, three people were asked to use six functions of the GUI with no prior instruction. They succeed in the first try most of the time (16/18 trials), whereas for the 2/18 failures, when doing it again, they could do it in the second attempt. The GUI was considered easy to use for 3/3 users (Table V).

TABLE V
NUMBER OF USER ATTEMPTS FOR EACH FUNCTION IN GUI

ID	*Add Springs*	*Add Damping Areas*	*Add Rigid Wall*	*Add Rough Wall*	*Add Facilitate Area*	*Export Environment*	*Easy to use*
1	One	One	One	One	One	One	Yes
2	**Two**	One	One	One	One	One	Yes
3	One	One	One	One	**Two**	One	Yes

In the first haptics experiment, six participants were asked to feel different types of environments (with visualization). All of them reported feeling the environment as intended. This indicates the success of force control to simulate the environment. Especially for the “free space," all of the participants reported it to be “very smooth," showing that the compensation of the friction and the dynamic model for the motor movement were successful.

In the second haptics experiment, when asked to guess the number of springs, 2/3 participants could only identify ¾ springs in the free space. With visualization, they could fully locate all the springs. This emphasized the importance of visualization for haptic devices. When asked to order the stiffness of the spring, 2/3 participants misclassified the strongest and second strongest spring (Table VI). This could be explained by the fact that these springs operated at our upper force limit and caused confusion.

TABLE VI
USER HAPTIC EXPERIMENT

ID	Guess number/ simulated number	Order the stiffness of springs			
		1st	2nd	3rd	4th
1	**3/4**	1st	2nd	3rd	4th
2	**3/4**	**2nd**	**1st**	3rd	4th
3	4/4	**2nd**	**1st**	3rd	4th

**Users were asked to feel the workspace with four simulated springs and guess the number. Later, they are asked to fell and order the spring stiffness.*

IV. DISCUSSION

Regarding controlling a three-DOF robot, the project successfully built a task space controller using a Jacobian-based approach. We found that adding PID control to the task space significantly increased precision.

With respect to environment generating programs, we have succeeded in building user-friendly GUI as well as making it possible for developers to easily add more environments. Regarding the haptics device, our method is a viable method for creating a haptic device. We successfully create scalable, modifiable, and stackable environments. The fact that the $f_{simulated}$ calculation is done at 529Hz while the hardware is limited to 40Hz, Hz indicates that a lot more complex environmental calculations can be added without affecting the performance. The length of the channel vector could be extended more if new information needed to be stored. The resolution for our environment is 1mm, and it could be even more precise by increasing the number of pixels used to store the environment. Currently, the environment requires $300 \cdot 600 \cdot 6 \cdot 64 = 69.12e6\ bits = 8.24\ MB$, which is easy to scale up as the RAM is normally in the GB range. Different techniques, like loading only a portion of the environment around the current position, are also possible if a finer or larger environment simulation is needed.

When generating the force, the haptics device demonstrated the capability to account for 83.33% of the variability in the desired force. As the force controller we used is an open system, the precision promises to be significantly increased if we add a force sensor at the end effector and use a close-loop control structure. The upper limit of the force that can be generated in our device was estimated in advance and used to guide the parameters when building the environment generating program. Future haptic device developers should be aware of their upper force limit in advance, as it is proven in the results that the haptics performance is limited around their upper limit. During hardware selection, an upper force limit should be considered to select those suitable for their purposes.

When discretizing the environment, the $f_{simulated}$ and $i_{simulated}$ were also discretized, risking the user having a serrulate perception. This was approached by choosing an appropriate haptics resolution in our design. The minimum distance that a person can discriminate between two points is at most 2mm, even for the tip of the index finger [23]. The size of our environment tensor is 300x600x6 for a 300x600mm, giving our device a haptics resolution of 1mm in both directions, which, we believe, is appropriate for human tactile perception resolution. The update frequency of the haptics device is 40Hz, therefore, to be able to feel every pixel of the environment, the speed limit of the end effector should be $1mm \cdot 40Hz = 40mm/s$. In our experiment, when the researchers tried their best to move the end effector the fastest, they could only reach 130mm/s (468km/h) when there was only free space and 50mm/s (180km/h) when there were obstacles in the environment. With 50mm/s speed, theoretically, the researcher only misses at most 1 pixel. And in practice, the users moved significantly lower than that (mostly less than 5mm/s), making it very easy for the haptics device to render every millimeter of the environment for the user to feel. This, we believe, is one of the successes in our design, and the mathematics mentioned could be used as a framework for future haptics devices if the developer decides to discretize the environment.

Our simulation method, however, only works for static environments. If one needs to simulate a dynamic one, where environment elements continue to change over time, it will require them to update the tensor in every iteration or have to try another method for simulating the environment. This, without doubt, will significantly increase the computational workload for the Raspberry Pi and might not be suitable.

The visualization is also a success, as the frames are updated at 18Hz. Given that the that the sampling rate of the human eye is 25Hz, the current visualization could be considered real-time. However, we acknowledge that there is room for the visualization to improve up to 25Hz. The current visualization was performed using Matplotlib, which is mainly used for data visualization; therefore, using other packages compatible with ROS might offer a higher update rate. Additionally, the hardware market offers a wide range of options that surpass the computational power of the Raspberry Pi, which could be used to significantly increase performance.

The findings of the user experiment conducted on the device underscore the significance of visualizations. Given that the human visual system is a highly vital sensory system, it is recommended that, in future system development involving the implementation of a feedback loop to enhance precision, visual feedback be prioritized over haptic feedback.

Last but not least, in our device, the haptic simulation was only focused on a single point on the end effectors. In real usage, collisions would occur for all of the robot links. A similar method for simulating the environment could be used. Math could be used to calculate the impact of each pixel on the whole arm and add them together. This is more mathematically heavy and was not experienced on our devices.

APPENDIX

Appendix.1. Formula for forward kinematics – the coordinate of the tips of arm 1, arm 2, and arm 3 stacked as 3 columns.

$$\begin{bmatrix} l_1 cos(q_1) & l_1 cos(q_1) + l_2 cos(q_1 + q_2) & l_1 cos(q_1) + l_2 cos(q_1 + q_2) + l_3 cos(q_1 + q_2 + q_3) \\ l_1 sin(q_1) & l_1 sin(q_1) + l_2 sin(q_1 + q_2) & l_1 sin(q_1) + l_2 sin(q_1 + q_2) + l_3 sin(q_1 + q_2 + q_3) \end{bmatrix}$$

Where $q = [q_1, q_2, q_3]$ represent the joint angles, $l = [l_1, l_2, l_3]$ represent three arm lengths.

Appendix.2. Formula for Jacobian J.

$$J = \begin{bmatrix} -l_1 sin(q_1) - l_2 sin(q_1 + q_2) - l_3 sin(q_1 + q_2 + q_3) & -l_2 sin(q_1 + q_2) - l_3 sin(q_1 + q_2 + q_3) & -l_3 sin(q_1 + q_2 + q_3) \\ l_1 cos(q_1) + l_2 cos(q_1 + q_2) + l_3 cos(q_1 + q_2 + q_3) & l_2 cos(q_1 + q_2) + l_3 cos(q_1 + q_2 + q_3) & l_3 cos(q_1 + q_2 + q_3) \end{bmatrix}$$

ACKNOWLEDGMENT

The project was part of Advanced Medical Robotics modules from King's College London. It was jointly contributed by Zhenguan Tu and Rui Han as Master of Science students, and supervised by Dr Lukas Lindenroth, Dr Alejandro Granados, and two teaching assistants Aya Zeidan and Peiyuan Sun.

REFERENCES

[1] A. Licona *et al.*, "Applications of Haptics in Medicine," 2020, pp. 183-214.

[2] T. Man *et al.*, "Surgical experience and different glove wearing conditions affect tactile sensibility," (in eng), *Heliyon*, vol. 8, no. 12, p. e12550, Dec 2022, doi: 10.1016/j.heliyon.2022.e12550.

[3] B. T. Bethea *et al.*, "Application of haptic feedback to robotic surgery," (in eng), *Journal of laparoendoscopic & advanced surgical techniques. Part A*, vol. 14, no. 3, pp. 191-5, Jun 2004, doi: 10.1089/1092642041255441.

[4] I. Camponogara and R. Volcic, "Integration of haptics and vision in human multisensory grasping," *Cortex*, vol. 135, pp. 173-185, 2021/02/01/ 2021, doi: https://doi.org/10.1016/j.cortex.2020.11.012.

[5] W. R. Sherman and A. B. Craig, "CHAPTER 6 - Interacting with the Virtual World," in *Understanding Virtual Reality*, W. R. Sherman and A. B. Craig Eds. San Francisco: Morgan Kaufmann, 2003, pp. 283-378.

[6] V. Hayward, "Haptics: A Key to Fast Paced Interactivity," in *Human Friendly Mechatronics*, E. Arai, T. Arai, and M. Takano Eds. Amsterdam: Elsevier Science, 2001, pp. 11-16.

[7] M. Morino, L. Pellegrino, C. Giaccone, C. Garrone, and F. Rebecchi, "Randomized clinical trial of robot-assisted versus laparoscopic Nissen fundoplication," (in eng), *Br J Surg*, vol. 93, no. 5, pp. 553-8, May 2006, doi: 10.1002/bjs.5325.

[8] C. P. Delaney, A. C. Lynch, A. J. Senagore, and V. W. Fazio, "Comparison of robotically performed and traditional laparoscopic colorectal surgery," (in eng), *Dis Colon Rectum*, vol. 46, no. 12, pp. 1633-9, Dec 2003, doi: 10.1007/bf02660768.

[9] O. A. van der Meijden and M. P. Schijven, "The value of haptic feedback in conventional and robot-assisted minimal invasive surgery and virtual reality training: a current review," (in eng), *Surgical endoscopy*, vol. 23, no. 6, pp. 1180-90, Jun 2009, doi: 10.1007/s00464-008-0298-x.

[10] J. T. Hamdi, S. Munshi, S. Azam, and A. Omer, "Development of a master–slave 3D printed robotic surgical finger with haptic feedback," *Journal of Robotic Surgery*, vol. 18, no. 1, p. 43, 2024/01/18 2024, doi: 10.1007/s11701-024-01819-8.

[11] D. Feygin, M. Keehner, and R. Tendick, "Haptic guidance: experimental evaluation of a haptic training method for a perceptual motor skill," in *Proceedings 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. HAPTICS 2002*, 24-25 March 2002 2002, pp. 40-47, doi: 10.1109/HAPTIC.2002.998939.

[12] A. Gutiérrez-Fernández, C. Fernández-Llamas, A. M. Vázquez-Casares, E. Mauriz, V. Riego-del-Castillo, and N. W. John, "Immersive haptic simulation for training nurses in emergency medical procedures," *The Visual Computer*, 2024/01/24 2024, doi: 10.1007/s00371-023-03227-9.

[13] P. de Zulueta, "Touch matters: COVID-19, physical examination, and 21st century general practice," (in eng), *Br J Gen Pract*, vol. 70, no. 701, pp. 594-595, Dec 2020, doi: 10.3399/bjgp20X713705.

[14] M. Kelly, W. Tink, and L. Nixon, "Keeping the Human Touch in Medical Practice," *Academic Medicine*, vol. 89, no. 10, 2014. [Online]. Available: https://journals.lww.com/academicmedicine/fulltext/2014/10000/keeping_the_human_touch_in_medical_practice.2.aspx.

[15] A. Tzemanaki, G. A. Al, C. Melhuish, and S. Dogramadzi, "Design of a Wearable Fingertip Haptic Device for Remote Palpation: Characterisation and Interface with a Virtual Environment," (in English), *Frontiers in Robotics and AI*, Original Research vol. 5, 2018-June-12 2018, doi: 10.3389/frobt.2018.00062.

[16] S. A. Bowyer, B. L. Davies, and F. R. y. Baena, "Active Constraints/Virtual Fixtures: A Survey," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 138-157, 2014, doi: 10.1109/TRO.2013.2283410.

[17] H. Y. K. Lau and L. C. C. Wai, "A Jacobian-based redundant control strategy for the 7-DOF WAM," in *7th International Conference on Control, Automation, Robotics and Vision, 2002. ICARCV 2002.*, 2-5 Dec.

2002 2002, vol. 2, pp. 1060-1065 vol.2, doi: 10.1109/ICARCV.2002.1238570.

[18] D. E. Whitney, "Historical Perspective and State of the Art in Robot Force Control," *The International Journal of Robotics Research*, vol. 6, no. 1, pp. 3-14, 1987/03/01 1987, doi: 10.1177/027836498700600101.

[19] "Anti-windup Strategies," in *Practical PID Control*, A. Visioli Ed. London: Springer London, 2006, pp. 35-60.

[20] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[21] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357-362, 2020/09// 2020, doi: 10.1038/s41586-020-2649-2.

[22] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90-95, 2007 2007, doi: 10.1109/MCSE.2007.55.

[23] S. Y. Won, H. K. Kim, M. E. Kim, and K. S. Kim, "Two-point discrimination values vary depending on test site, sex and test modality in the orofacial region: a preliminary study," (in eng), *J Appl Oral Sci*, vol. 25, no. 4, pp. 427-435, Jul-Aug 2017, doi: 10.1590/1678-7757-2016-0462.