Remote Feelings

Xinying Hu, Aymeric Wang, Adam Curtis
{xinying_hu, wang_aymeric, adamcurtis}@berkeley.edu

ABSTRACT

Telesurgery allows doctors to perform surgery without being physically present with the patient. In addressing worldwide shortage of surgeons, telesurgery aims to offer surgery assistance despite geographic barriers. However, current surgeon training relies heavily on touch to perform precise actions, which is why telesurgery usually bases itself on haptic feedback. **[1]**

Our project aimed at a similar use case of haptic feedback. We wanted to design hardware to allow a user to remotely control a robotic arm and gripper while feeling haptic force feedback based on forces sensed by the gripper.

1 METHODOLOGY

1.1 CHOICE OF HARDWARE

This project consists of two main hardware components: the haptic force feedback glove and the robot. The glove represents the bulk of the hardware complexity and design work, while the gripper exists to illustrate the abilities of the glove. Creating the robot arm also allowed us to explore the capabilities of computer vision (CV) in tracking the human body. This allowed us to include an additional motion based on the user's elbow, without building any extra hardware.

Figure 1: **Overall view of Remote Feelings**

1.1.1 Glove hardware description

The glove design features many components with the ESP32 microcontroller (MCU) at the center. A custom PCB bridges the electrical components with the sensors and actuators. Actuation was achieved with modified servo motors to expose control of its H-Bridge driver and to output encoded rotation used to track finger positions. We also measure the force applied to each finger using force sensing resistors (FSR). The glove main body is strapped to the hand to contain the PCB, and support the servos. Each servo and FSR are held together by a custom bracket, which applies a pre-load force on the FSR, allowing us to detect forward and backward finger movements. Standard shoulder screws create rotating joints on the servos for comfort. Accordingly, the thumb can move freely thanks to a two-degrees-of-freedom articulation.

1.2 SYSTEM ARCHITECTURE

Figure 2 below illustrates how components interact with each other. ADC force measurements are passed to servos to simplify communication. These measurement values are linearly mapped to a duty cycle for the PWM signals of the servos.

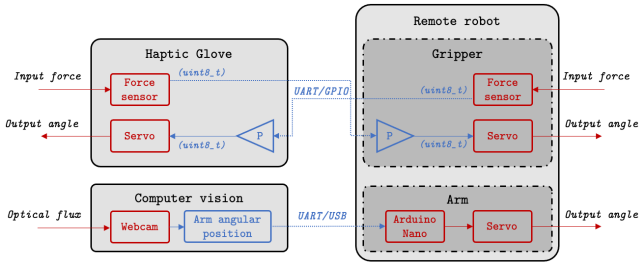

Figure 2: **System Architecture of Remote Feelings. Communication symbolized in black, software processings in blue and hardware in red.**

1.3 VISION CONTROL

Tracking elbow rotation is achieved by recognizing human body landmarks and calculating the moving angles between them. We achieve this using the MediaPipe framework **[2]**. We illustrate how we exploit these landmarks to track elbow rotation in

Figure 3. We map the arm angle to the arm angle of the robot servo using the equation:

$$Robot\ arm\ angle = 180 - User\ arm\ angle$$
$$= 180 - arccos\left(\frac{SE \bullet EW}{|SE||EW|}\right)$$

Considering the angular range of the robot arm we cap angular positioning to a [40, 120] degrees interval.

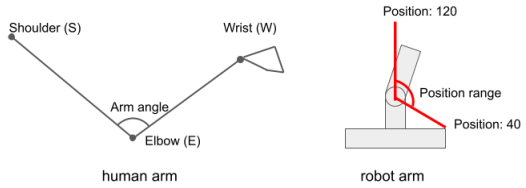

Figure 3: **Matching user arm angle to robotic arm angle using landmarks**

1.4 SYSTEM MODELIZATION

1.4.1 Gripper/Glove force messages parsing logic

Our code uses a similar compilation logic as Arduino, meaning our processes are supposed to be atomic. This helps us modelize the communication of force measurements using the following Finite State Machine (FSM)

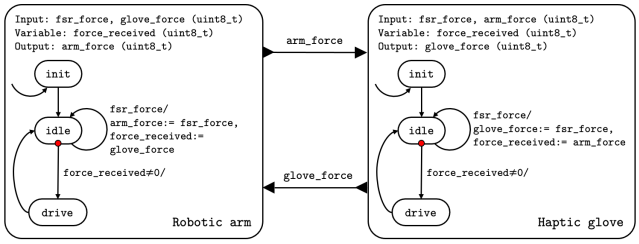

Figure 4: **FSM modeling the force message parsing between the robotic arm and the haptic glove**

Note that our model is considered *asynchronous* which allows both machines to operate independently and gain in responsiveness. We introduced a `fsr_force` local input corresponding to the ADC measurement of FSR sensors. The states in the FSM traduce the following behavior:

- `init`: Is the initialization of the program. It is immediately skipped
- `idle`: Waits for a non null force difference (`force_received`) between FSR measurements and inputs from the UART communication.
- `drive`: If `force_received` ≠ 0, drive the servos according to control strategy.

1.4.2 Control flow of the CV algorithm

The computer vision aspect of the project is described in **figure 5** as a control flow graph of the code.

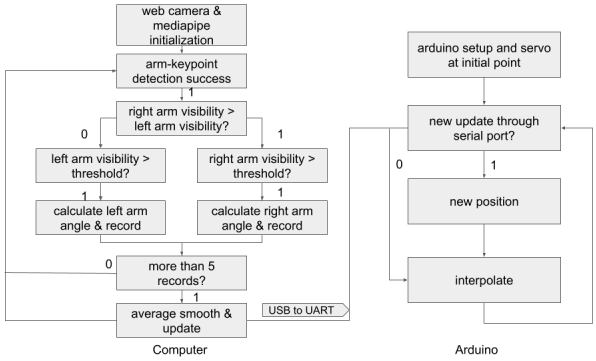

Figure 5: **Control logic of the vision-controlled robot arm**

1.5 COMMUNICATION

1.5.1 Gripper/Glove force measurements

We made the gripper and the glove communicate their force measurements using UART over an ethernet cable. Measurements were processed thanks to their respective ESP32. The symmetry of this design allowed our software to be quite analogous for both platforms which conveniently abstracted the software implementation.

1.5.2 Synchronization for vision control

The CV algorithm runs on an external laptop that sends the user's elbow angle via USB-UART to an Arduino Nano. Timing and synchronization need to be maintained between the two control units in order to achieve real-time control. On each iteration, the CV algorithm computes the user's elbow angle with higher visibility and averages it on five sample points before sending the value to the Arduino. Conversely, the Arduino uses a higher refresh rate to detect new elbow angle updates and command the robotic arm.

2 PERFORMANCE

We were able to design a functioning haptic force feedback glove controlling a single-jointed robot arm. This way, estimating the performance of our solution falls quite naturally.

2.1 STRENGTHS

2.1.1. **“Hacking” the servo motors to get rid of PID**

The first version of our glove was presenting some oscillations that made the glove unusable. Our first approach was to design a Proportional Derivative (PD) corrector that would offset this behavior. Doing this was conversely complexifying our software implementation as well as increasing the responsive time of our glove. One hack was to modify the servos so we could interface their H-bridge. This allowed us to directly control torque on the motor, which correlates with force instead of controlling motor angle, which is related through a double integration to the finger force. As a result, we were able to linearly map force measurements from the robot to motor power output on the fingers using a simpler proportional (P) controller. This hardware trick allowed us to get better performance of the glove in free movement as well as during force exertion: it was much more responsive and was not oscillating anymore except in one edge case when users attempt to apply a very light force, which will be important to correct in the future.

2.1.2. **Price and accessibility**

Compared to the similar haptic glove present on the market, our solution has the benefit of being open-source [3] and less expensive than most haptic gloves. The most popular one is designed by SenseGlove and addresses professional markets that use VR. It tops $5000 dollars [4] when we estimate our solution to cost around $300 considering the fact we used calibrated shoulder screws ($5/ea), designed a custom PCB, used expensive FSRs ($10/ea) and metal geared servos ($6/ea). Being able to print inexpensive 3D parts lowered the cost of hardware, implying an owner of a 3D printing machine might replicate the glove for less even than our estimation if electromechanical parts’ cost get further optimized.

2.1.3. **Wired based communication**

The use of a UART protocol proved to be a good decision. Considering the different wireless protocols offered by ESP32 platforms we were first considering sending messages over WiFi or BLE. However we quickly realized that wired communication would guarantee a sufficient baudrate. Moreover, UART was picked since we knew UART was well supported by our embedded platform.

2.1.4. **Scalability with software portability**

Another advantage of our solution is its scalability thanks to its portable software development. We coded the program to control a simple robot arm in C/C++, which is supported by most embedded platforms. However nothing - if not time - could prevent us from controlling a professional industrial arm that is controlled by its own embedded platform and powered by its own power source. However, the UART will have to be translated to TCP/IP, which can be done using a bridge. **[5]**

2.2 WEAKNESSES AND FUTURE WORK

2.2.1. **Mitigating asynchronization of UART**

We ran into several problems using UART, one of which being the asynchronous property of UART. Because our solution presents concurrent processes, both our CV algorithm and our communication were suffering before fix:

- For a whole week, the robotic arm was not moving despite the commands sent by our CV algorithm
- Similarly, the force messages we communicated between the robotic arm and the glove were not properly read by our instances. Every second we were seeing either partial or chunked information.

We mitigated this issue by polling the UART buffer to check for a beginning character and waiting for the message terminating character before passing that message as usable value to the rest of the system.

Experimenting showed that ESP32 microcontrollers thread UART reading processes to improve performance. We ended up with memory segmentation errors where one thread was trying to access data that was previously tossed out of the UART buffer. We fixed this issue by either implementing a quick “try-catch” fix to make the

microcontroller disregard these errors or by adding a significant delay for the whole message to be received. A more secure and optimal approach would be to use timer interrupts so the UART will only be read by one thread at a time.

2.2.2 **Non-linearity in the mechanical design**

Despite the upgrade to the servo motors, we still faced a nonlinear control problem. As the fingers curl inward the force applied to the fingers due to the force of the motor changes dramatically in magnitude and direction. **Figure 6** shows these nonlinearities in a free body diagram.

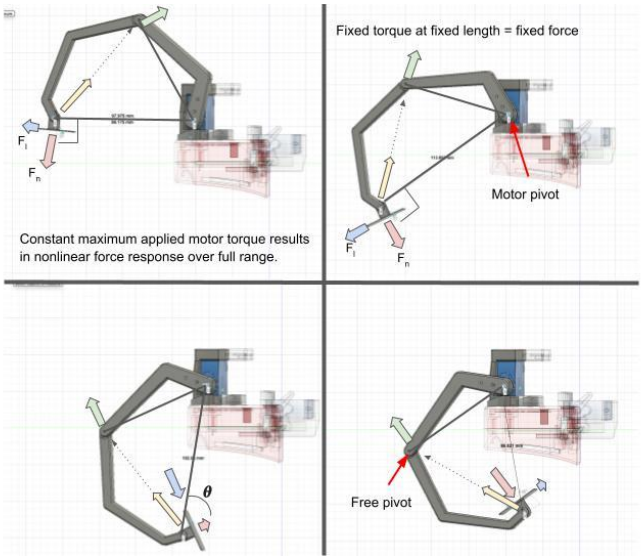

Figure 6: **Nonlinear force response from a constant motor torque across the full range of finger movement.**

This implies that the controls' performance will always be deteriorated when the fingers are at their maximum curl inward. One mitigating solution is to adjust the torque on the motor depending on the curl of the fingers to create a perceived linear force.

2.2.3 **Improving sensitivity of our force sensors**

Another issue arose in the sensitivity and range of the FSR. We chose the form factor FlexiForce A101, which can sense 0.25 - 4 lbs of force. The resistance over this range is not linear and using a voltage divider circuit to measure it further increased these nonlinearities. This method for measuring the FSR also limited dynamic range.

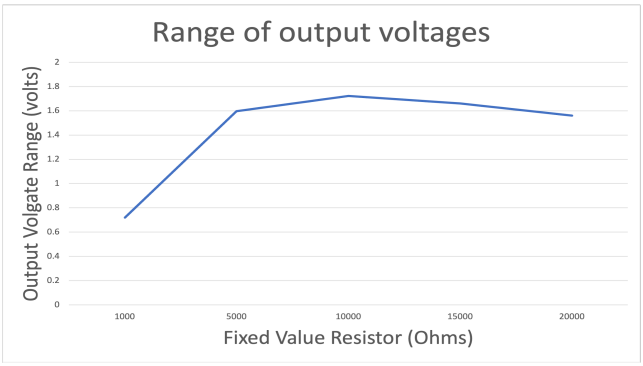

Figure 7: **Dynamic range of voltage outputs for force sensing resistors coupled with a 1k - 20kΩ resistor from in a voltage divider circuit.**

Using a 10k fixed value resistor for the voltage divider optimized the dynamic range of the FSR voltage divider circuit to a total of 1.75 volts as shown in **Figure 7**. We could achieve better linearity and about 80% more dynamic range by using a non-inverting op-amp circuit as shown in **Figure 8**.

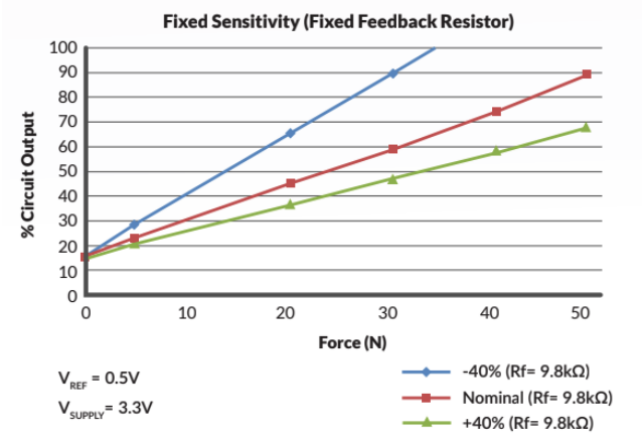

Figure 8: **Output voltage for force sensing resistors in a non-inverting op-amp circuit.**

ACKNOWLEDGMENTS

We would like to thank professor Sanjit A. Seshia for the opportunity to do this wonderful project and all the course materials covered in course EECS 149/249A which not only helped us increase our knowledge and skills but also better tackle the technical problems during the project. We would also like to express thanks to the GSIs and all the course staff for the advice and help for the project and labs. We would

not be able to finish our project without your guidance and support!

REFERENCES

[1] Telesurgery: Past, Present, and Future, Paul J Choi, Rod J Oskouian, R. Shane Tubbs, Cureus. 2018 May; 10(5): e2716. Published online 2018 May 31. doi: 10.7759/cureus.2716,
[2] https://mediapipe.dev. last visited: 12/17/2021
[3] https://github.com/BerkeleyCurtis/EECS249_HapticGlove,
[4] https://www.senseglove.com/product/nova/, last visited: 12/17/2021
[5] https://documentation.help/Microchip-TCP.IP-Stack/UART-to-TCP_Bridge.html, last visited: 12/17/2021