Building a Robotic Platform for Recreating Realistic Human Movement in Real Time

Elliot A. Gorman

[a] *McMaster University*
[b] *Assigned Laboratory: MAEDA Lab Graduate School of Information Science and Technology*
[c] *Supervisor: Professor Taro MAEDA Associate Professor Masahiro FURUKAWA*

Abstract

Tele-existence is the concept of using robotic technology to be 'remotely yet physically present'. This involves two physically distant robots: a master robot controlled by a human, and a slave robot which replicates the master robot's actions and can give relevant feedback to the human. While the design of such systems is heavily domain-dependent, there are several key requirements often necessary. Namely, the human controls must be immediately intuitive, and the system must communicate between the two sides as fast as possible. Formal training must not be required to use the system, and noticeable delay between the master and slave will shatter the illusion of being physically present. We are building a tele-existence system utilizing commercially available parts to transmit human arm, and later, head movement. Currently, the slave robot uses assembly delta robots to model arm motions, and the master robot — also a delta robot — takes human arm movements as input. Our goal is to increase the transmission frequency between the master and slave robots, along with minimizing the transmission latency to near real-time. To compete with the slave robot manufacturer's designated robot controller, which is too slow for our purposes, we implement our own slave robot controller utilizing consumer microcontrollers and motor drivers. The results have been measured independently of the slave robots' physical movement, and thus are purely theoretical. The transmission frequency is greatly increased, over double the original amount. Furthermore, the transmission latency is decreased. Theoretical accuracy of movement is retained to a high degree. Although there are several improvements to be made to the custom controller (e.g. inverse kinematics tuning, calculation code optimization, and transmission format optimization), a marked increase in performance is gained over the original controller.

Keywords: Tele-existence, Human Interface Devices, Delta Robots, Microcontroller Development

1. Introduction

The concept of tele-existence is an evolution of tele-presence. Tele-presence, a more primitive version of 'transmitting' the human self across space utilizing technology, can be thought of as a video call using one of the many common telecommunication applications today.

For example, consider the following setup. In one location, there is a 'master system' with a human and a camera, capturing video of the human's body and sending it to a 'slave system' in another location. This other system, equipped with a monitor displaying the video it receives from the master — and a camera also sending back video — allows the human to be 'present' at its location. This is the bare minimum for transmitting presence since the human can only mentally engage with the other location.

With tele-existence, this setup is extended to now include the transmission (and replication on the other end) of at least one component of the physical, in-the-flesh, existence of the human. Components often transmitted are the arms, upper torso, and head.

This change from tele-presence requires replacing (or supplementing) the master's camera with a motion capture device that can convert motions of the component(s) to data that can be transmitted to the slave. Furthermore, the slaves' monitor must

be replaced (or supplemented) with a device that can replicate the component(s)'s motion according to the data sent from the master.

Having such a broad definition, tele-existence encompasses many types of robotic systems and is used in many different applications. For example, a system that transmits arm movement may be used in assembly or inventory stocking, but it could also be used in surgery or other medical procedures.

Common among most implementations of tele-existence is the need for non-intrusive controls and as little latency as possible.

The controls that the human uses to have their existence captured by whatever the input device is must be 'transparent'. That is, the human must make little-to-no accommodations to their movement for the sake of the input-capturing device. If the human makes compromises to their movements to account for the input device, then the human is not naturally and truly existing in another location — they are instead tele-operating a distant machine.

Similarly, unless the application directly calls for it, it is important that the human's movements be transmitted and replicated as fast as possible. Significant delay will quickly deteriorate the usability and may render it impossible for the system to do its task. For an example, consider the case where the slave robot is positioned on an assembly line, and needs to keep up with the flow of parts as fast as possible. Noticeable delay would make it extremely difficult for the human operator to maintain coordination.

2. Objective

We wish to create a general-purpose and generic tele-existence system in which the physical components transmitted are both arms (including wrist movement) and the head.

General purpose meaning that there is no current application for the system — rather it should be easy to modify the setup to fit a relevant task — and generic meaning that we want to use only industrial, already manufactured robots. The advantage of this is that, unlike most other tele-existence projects which create the slave and master systems from scratch, there is significantly less setup time, and greater flexibility since the robot has not been tuned and developed with a specifc purpose in mind.

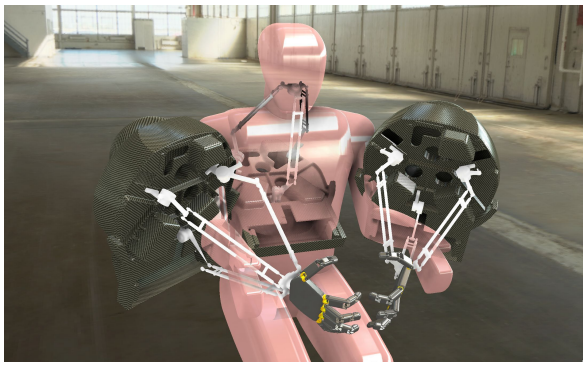

Figure 1: Prototype rendering of the slave side of the general purpose tele-existence system

The scope of this paper is limited to implementing tele-existence over one arm (excluding wrist rotation) in this system, keeping in mind the latency restrictions. This serves as a proof-of-concept for the rest of the system, and can hopefully be easily extended to include the other components.

3. Method

The master device for an arm in this system is a sigma.7 haptic device from Force Dimension, and the slave system is a FANUC Robot M-1iA 0.5AL. Note that, for greater simplicity, both robots are delta parallel types.

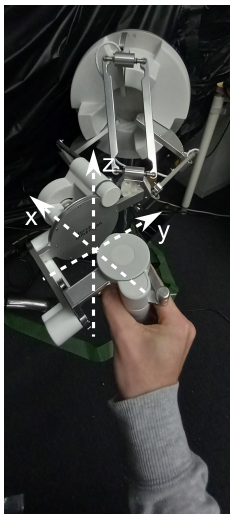

Figure 2: Force Dimension sigma.7 with coordinate system and control method

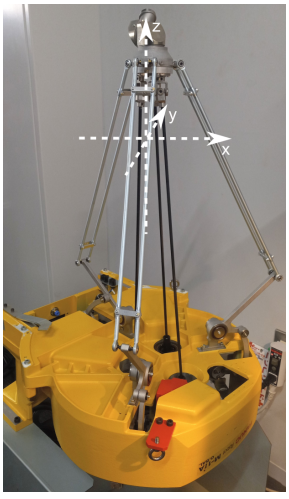

Figure 3: FANUC Robot M-1iA 0.5AL with coordinate system

The master system allows for transparent, easy control via a gripper where the hand can be placed and then moved around, friction-free. The position of the gripper can then be accessed via the manufacturer-provided API, on an x86˙64 Linux machine in our case.

The slave system can be controlled via the manufacturer supplied controller, a FANUC Robot R-30iB/R-30iB Mate controller. This controller allows for local network communication (e.g. over UDP), so an example setup would be to have the computer running the sigma.7 API communicate with the FANUC controller over the network.

However, after implementing this setup and performing measurements, the resulting latency and transmission frequency is too slow for our goals. Therefore, to increase these two measurements, we sacrifice some generality to introduce our own controller for the slave robot. Not all is lost however, as the custom controller is built with an easily accesible, off-the-shelf Espressif Systems ESP32 microcontroller.

To introduce our own controller to the system requires direct connection to each of the three motors in use in the slave robot. With reverse-engineering efforts, it was found that the motors' encoder output (A, B, Z lines) could be accessed via through-holes in the encoder's PCB, located at the back of each motor.

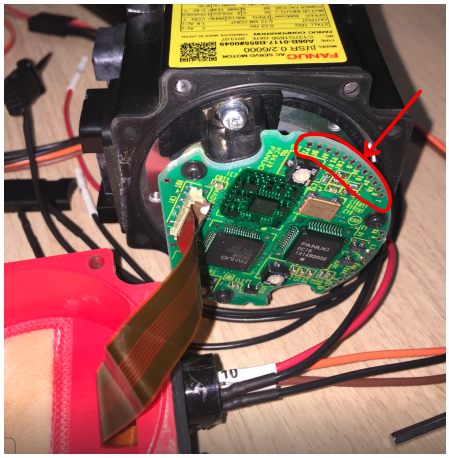

Figure 4: Encoder status through-holes on PCB (pointed to and circled in bright red), located at back of motor

On top of motor control, the FANUC controller also does various other computations which we must implement in our custom circuit. Namely, we must implement inverse kinematics. Since the inclusion of a computer is necessary for the sake of using the sigma.7 API, we can take advantage of its faster speed to share the workload and have it perform the mathematics of the inverse kinematics.

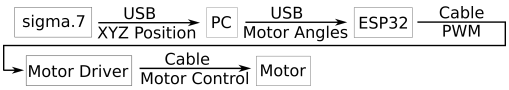

Figure 5: Flow chart of information from master to slave (sigma.7 to motors)

As in figure 5, the first step is the transmission of xyz positions from the sigma.7 to the USB connected computer (via the sigma.7 C++ API). Upon receiving each positional update, the computer then performs the necessary inverse kinematics so as to calculate the angle each motor needs to be at. After this, the computer can send the motor angles over serial via the USB to UART adapter on-board the ESP32. Next, the ESP32 can generate the necessary PWM to rotate the motors to the specified angle, and send them to the motor driver, which commits the final rotations.

For our motor driver, we use a Servoland MOVO2 SVFM2-E6-DSP for each motor, resulting in a total of three.

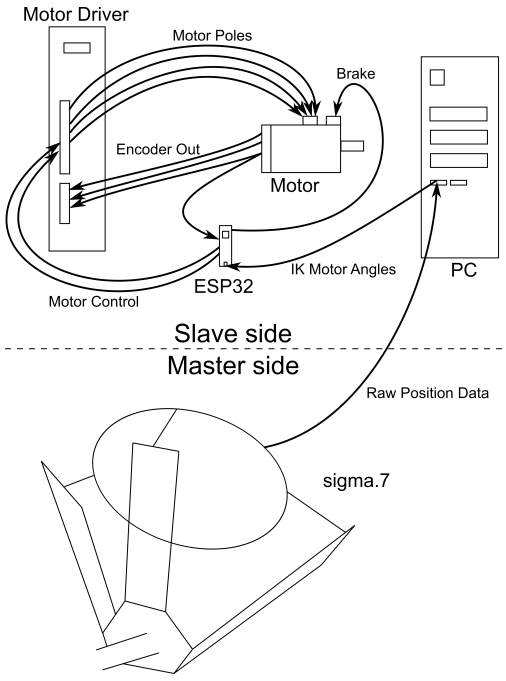

Figure 6: Outline of circuit from master to one of the three motors used in the slave

Currently, as can be seen in figure 6, the setup

includes a single ESP32. Since motor control on the ESP32 side consists of three GPIO channels — one for the motor speed, one for the direction, and one input for the Z encoder pulse — the ESP32 can handle all three motors. However, were more motors in use, this setup would probably not extend well due to the limited resources of the ESP32.

With all the hardware in place, the first challenge is to convert from the sigma.7's coordinate system (set by the manufacturer) and the custom coordinate system for the FANUC. A custom coordinate system is created for the FANUC because it is convenient for the inverse kinematics calculations.

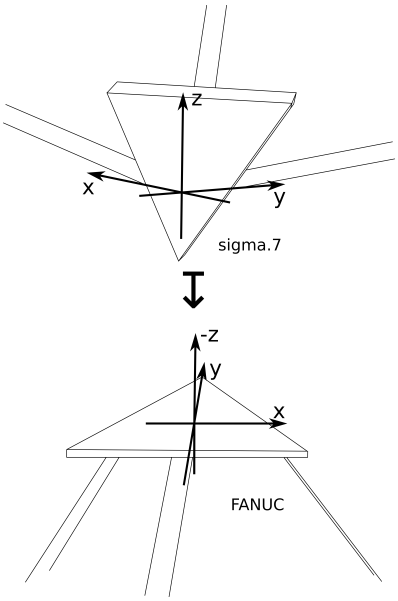

Figure 7: Differing coordinate systems between sigma.7 (master) and FANUC (slave). The triangle represents the end effector

Fortunately, the direct conversion is quite simple, requiring only two matrix multiplications: a 45-degree rotation about the z-axis, and a swap of the z and y axes.

$$\vec{P}_{\text{FANUC}} = \vec{P}_{\text{sigma.7}} \cdot \begin{bmatrix} \cos(45^\circ) & -\sin(45^\circ) & 0 \\ \sin(45^\circ) & \cos(45^\circ) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & (-1) & 0 \end{bmatrix} \tag{1}$$

Due to the different physical lengths of the arms, slight scaling needs to be applied to the resulting output position. However, the constant values used for scaling are experimentally derived and as result, the inverse kinematics algorithm is a partial function.

For the implementation of the inverse kinematics algorithm, a geometrical approach has been taken, for simplicity and to reduce computational complexity. A heavily simplified and abstracted model of the FANUC delta robot is used, based on the geometry presented in [1].

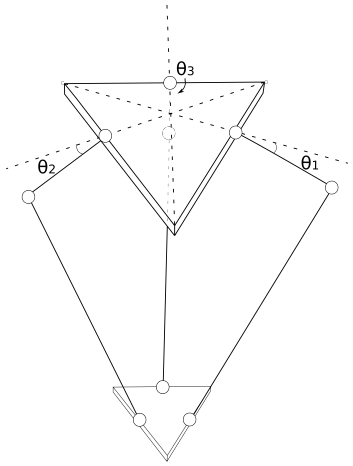

Figure 8: Abstracted model of FANUC (slave) base and end effector

The triangular prisms, though not completely true-to-life, allow for the manipulation of basic trigonometry rules, resulting in a simple implementation and less work done per each positional update.

Each time the computer connected to the sigma.7 receives a positional update (over USB and captured by the API), the inverse kinematics algorithm is run three times, once for each motor.

Unfortunately, the project timing constraints did not allow for the physical placement of the motors (which were extracted for development) inside the FANUC body. Therefore, all tests and measurements mentioned in the results section remain strictly theoretical.

4. Results

Various different types of tests were performed to benchmark the performance of the custom controller. The two measurements of interest were the transmission latency and speed. The transmission latency refers to the total time elapsed from the

sigma.7 API updating the current position, to the last of the three motors' pulses being generated. The transmission speed refers to the number of positional updates successfully completed (i.e. all pulses generated) per second.

The first test performed was to count each positional update (after all pulses were generated) in the ESP32's code, per a 10 second interval.

The immediately apparent phenomenon is that this speed is directly proportional to the amount of acceleration featured in the arm movements over the 10 seconds. That is, the faster the arm movements were, the less full updates completed. This makes intuitive sense, as a larger error in the motors angles means more iterations of pulses, however, such a direct correlation was not expected. This may be due to the particular ESP32 model in use — the RMT peripheral on board (which was used for PWM generation) lacked the hardware ability to send out exact numbers of pulses. As a result, a software solution was used, in which a single pulse is generated in a loop (which iterates the exact number of times needed).

With slower, less accelerative motion, the result measured by this test is that on average, 115-125 updates are possible per second.

For latency, a slightly different test was used. The master and slave system were run as regular until 500 positional updates were reached, after which point the test stops. During the run, for each positional update, the master attached the number of updates so far and a timestamp (using the standard `gettimeofday()` function) to the position, and saved it in RAM, before writing to a file at the end of all transmissions. The slave did the same, though needed to connect to an SNTP server first via the WiFi card.

Importantly, there was no synching of the two clocks, and so the results of the test are not concrete, though they do paint a good enough picture to get an idea of the latency. The mean latency (difference between the timestamps recorded by the master and the slave) is 105ms, with the standard deviation (from the entire population of timestamp differences) being 270ms.

Finally, a third theoretical test was performed to confirm the functionality of the inverse kinematics algorithm. By feeding the output of the inverse kinematics to a forward kinematics algorithm, the accuracy of the inverse kinematics algorithm can be observed.

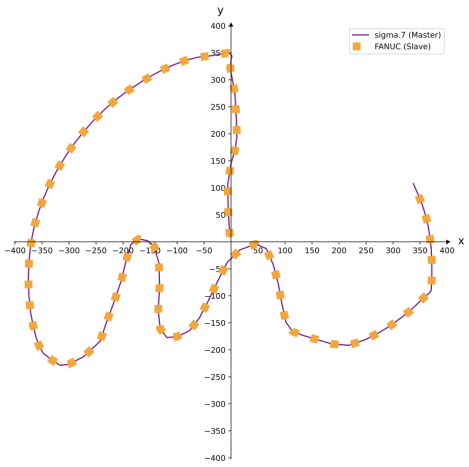

Figure 9: Comparison of sigma.7 position with forward kinematics output after being fed inverse kinematics output

5. Discussion

Despite the large variation during moments of rapid acceleration, the current controller manages to slightly outperform the FANUC default controller in latency and frequency, a majority of the time.

For frequency, on a sample run, the difference in mean frequency between the original and custom controller is roughly 76Hz.

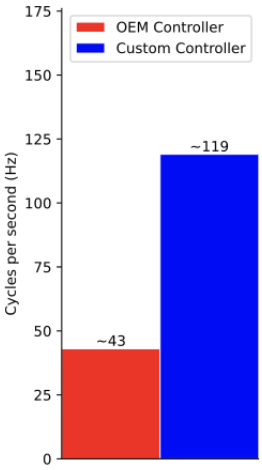

Figure 10: Comparison of transmission frequency between the FANUC controller and the custom controller on a sample run

However, during this run, acceleration remained rather low and it is assumed that, under extreme acceleration, the difference will be smaller.

For latency, on a sample run, the difference in mean latency is roughly 245ms.

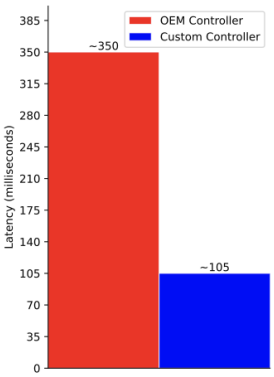

Figure 11: Comparison of transmission latency between the FANUC controller and the custom controller on a sample run

For an example of the heavy variation during acceleration, observe the scatter plot in figure 12, in which the latency between the sigma.7 and controller spikes up to as high as 1000ms during extreme acceleration, but remains minimal otherwise.

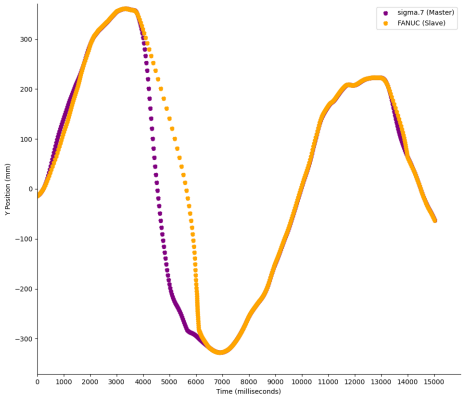

Figure 12: Latency between sigma.7 and custom controller on sample run

6. Conclusion

Based on the quick measurements and comparisons performed, it is evident that there are vast improvements to be made to the custom controller. With further optimization made to the kinematics calculations, as well as to the transmission communication, it may be possible to significantly reduce the impact of acceleration on the performance of the controller.

As it stands, the controller does present a considerable improvement over the FANUC controller, under the circumstances of little acceleration.

References

[1] Zsombor-Murray, P. J. (2004). *Descriptive Geometric Kinematic Analysis of Clavel's Delta Robot.* McGill University, Department of Mechanical Engineering, Center for Intelligent Machines, Canada