Typing Robot: Typing with a 4-Fingered Dexterous Hand using Motion Planning

1st Monica Chan
MIT EECS
Robotic Manipulation 6.4210
Cambridge, USA
mochan@mit.edu

2nd Cynthia Zhang
MIT EECS
Robotic Manipulation 6.4210
Cambridge, USA
zcynthia@mit.edu

***Abstract*—We investigated whether a robot can accurately type a provided message on a standard QWERTY keyboard. This project aims to simulate the robotic manipulation task of typing using motion planning via inverse kinematics. The robotic system is an Allegro hand end effector attached to a Kuka iiwa arm. The Allegro hand is analogous to a human right hand, with one less finger. The investigation of this question involves constructing a realistic keyboard simulation that can (1) simulate key presses, (2) detect key presses, and (3) display key presses. Following this, an inverse-kinematics based motion-planning approach is used to determine the joint angles required to press each key. The resulting system is capable of typing messages using multiple fingers with high accuracy.**

***Index Terms*—robot, typing, inverse kinematics, motion planning**

I. Introduction

Typing is a manipulation task that many humans complete every single day. Whether a human is using all of their fingers or just one, they are able to accomplish this task quickly and accurately. Being able to model this motion as a robotic manipulation task can help contribute to the field of making robotic hands more dexterous. While a robot that types is not in itself useful as an application, studying the motion can, for instance, contribute to the field of prosthetics. Typing is a motion that someone without a hand would have to be able to complete day-to-day, so studying the motion and paths for this application is valuable.

We anticipated challenges to arise from the high precision requirement of the robot hand. Keyboards are a relatively small, tight space with much room for error. Any inaccuracies will likely result in neighboring keys being hit. Being able to type accurately will be a great advancement for robots to perform high precision tasks, with potential applications in the industrial or medical fields. For example, high precision assembly tasks, or even surgeries could be made possible with dexterous hands that can perform tasks with high precision.

While human typing is typically accomplished using two hands, trying to model this in a robotics system would pose a large challenge. The Kuka iiwa arm and Allegro hand are large, clunky pieces of hardware. Fitting two on a relatively small keyboard would greatly complicate collision detection as the hands will collide very frequently. Furthermore, using only one arm is more resource efficient, as increasing to two arms provides limited improvements.

The project uses the Drake robotics library in Python (pydrake) for all simulation and robotic manipulation tasks. The simulation environment includes two tables on which rest the robot and the keyboard. The robot is the Kuka iiwa arm, on which is attached the Allegro hand end-effector. The keyboard is simulated using a long block to represent the base, and 26 smaller blocks to represent each key letter. A longer block represents the space bar. Two views of the set up can be seen in Fig. 1.

II. Related Work

Seong Ho Yeon, an alumnus of this class, completed a project of a robot that plays piano with a single robotic hand [1]. The piano playing robot raises some similar considerations as this project's design including the motion planning and force control, since a piano keyboard bears similarities to a QWERTY keyboard. Specifically, the project explores different hand positions used to press piano keys without disturbing keys around it, which relates to the task of pressing keyboard keys without disturbing other letter keys around it. This piano project also explores the transitions between different hand positions, which our project also aims to have. Additionally, this project was inspired by Yeon's project for how to simulate the keyboard by observing the method by which he generated each piano key and its spring-like nature [1].

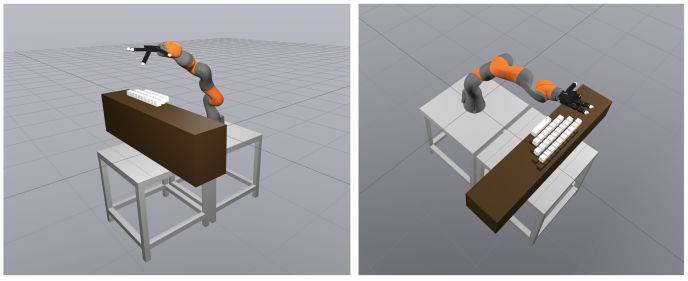

Fig. 1. Simulation arrangement. From the left, one can observe each element present: two tables, the kuka iiwa arm, the Allegro hand, and the keyboard. The right provides an aerial view of the keyboard layout.

A different paper discusses both characterizing and determining correct contact forces for fingers playing a piano [2]. It uses a switched controller to help determine the punching force of each key, and this dictates what types of problems are needed to solve for the contact force of the keyboard. This project's examination of punching force is simpler, as a typing on a keyboard only needs to register a click, and does not care about the lightness or heaviness of a click.

III. Simulation and Setup

A. Starting point

The project started from a class Deepnote notebook that included the robot and tables. The primary task was to figure out how to generate a keyboard into this environment.

B. Key representation

The process for setting up the keyboard involved first figuring out how to simulate a singular key, and then setting up multiple in a row setup like that of a real keyboard.

The first challenge was figuring out how to generate a single block within the simulation. Originally, the plan was to include existing .sdf models to generate a blocks. However, RigidBodies were used instead for simplicity.

The next step was to make the key spring-like. Once pressed, it should return to its initial state. Our initial approach was to either simulate a literal spring object, or to deform the block in a "squishy" manner that then bounces back to the initial form. Ultimately, the PrisimaticJoint and LinearSpringDamper functions were used to create a key. The key is defined by two blocks, stacked on top of each other, with a joint connecting the two pieces and a spring defining the nature of how the blocks move on this joint. This can be seen in Fig. 2.

The prisimatic joint allows for one degree of freedom motion in the Z axis. This range ensures that the key cap can only move up and down, similar to how real keyboard keys are constructed. The spring damper enables the bounce-back nature of the key.

In a linear spring damper system, the force can be modeled as such:

$$F = -k \cdot x - b \cdot v \tag{1}$$

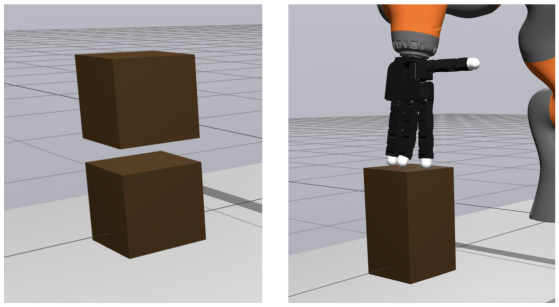

Fig. 2. Enlarged structure of a single keyboard key in simulation. On the left, the resting state of the key. On the right, the pressed state of the key. Once the robot hand is removed, the key will return from the pressed state into the unpressed state.

where k is the spring constant and b is the damping constant. The following are the values of the damper settled with in simulation:

- block mass: 0.1 kg
- free length: 0.2 m
- stiffness (spring constant) = 10 N/m
- damping = 1.2 Ns/m

We considered using force detection to determine whether a key was pressed. However, we ultimately decided to calculate position deflection to detect key presses, as described in section III-D. We are keeping the equation as an improvement for the future.

C. Generating a Keyboard

After figuring out how to generate a keyboard key, the next step was to generate the whole keyboard. We first needed to reduce the key size. We ultimately decided with cubic keys with side length of 0.04m, as this key size is slightly larger than the Allegro hand's finger tip, much like how human keyboard keys are slight larger than our finger tips.

To facilitate this process for all 26 letter keys, a helper function was written to create each key. This function is then called 26 times within a loop to build each key. The key positions are stored in a dictionary where we figured out the optimal spacing between rows and between keys. The spacebar is handled separately as it has different key dimensions than the letter keys.

D. Detecting a Key Press

A function was written to detect key presses. The function takes a key of interest and an initial position to perform a deflection calculation. If the current and initial z positions of a key have a large enough difference, a key detection event takes place, and the key name is printed to the console. Console output can be seen in Fig. 3. At every time step, the code iterates through all of the keys on keyboard and calls the detectPress method to detect both correct and erroneous key presses.

$$x_t^z - x_{t-1}^z \geq 0.01 \tag{2}$$

Equation (2) explains the detection of a press mathematically, where x_t^z is the distance between the top of the key and its base in the z-axis direction at time t, similar to x_{t-1}^z for time $t-1$.

We thought about how the force control needed for pressing a keyboard on an external keyboard versus a laptop would be

```
T  pressed!
H  pressed!
E  pressed!
spacebar  pressed!
Q  pressed!
U  pressed!
I  pressed!
C  pressed!
K  pressed!
spacebar  pressed!
```

Fig. 3. Example output of the system detecting key presses. From the console output, we can visually confirm the keys being pressed.

different, and so in the future using a force controlled method may be more effective for trying to type on different types of keyboards.

IV. Technical Approach

A. Mapping out Hand Joints

The Kuka iiwa arm in tandem with the Allegro hand gives, in total, 23 joints to manipulate. In order to understand how to manipulate the robot, a critical first step was to understand the joint mappings for the hand.

These joints are manipulated using an array of 23 values. The first 7 values set the arm joints. The remaining 16 joints are the hand joints. With experimentation, a list was developed that mapped each joint to an image of the part the joint moved to understand what each joint did. Some examples of these mappings can be seen in Fig. 4.

B. Pressing a single key

The first approach to get the robot moving at all was hard coding joint positions to gain a basic understanding of how the robot moves and how the joints relate to each other. When testing the functionality of the keyboard, hard coded joint positions were also used. This meant experimenting with various values for each of the joints to determine a rough joint arrangement that allowed the system to press and unpress a key. This allowed observation of the spring nature of the keys and ensure they are functioning as expected.

This basic hard-coding approach gave key insights into how to motion plan using the eventual inverse kinematic controller. The hard-coding approach involves first hovering over the desired key, and then lowering the hand onto the key to press, then returning to the hovering position. This approach ensures that the robot hand will not hit other keys during its trajectory into the pressed position.

When planning the motion to press a single key, the process was simplified to using just the index finger to push the button with the other fingers folded in. After that functionality was working, we implemented motion planning using multiple fingers with the basic one-finger framework. We experimented with two main hand orientations: one with only the index finger extended in a pointing-like manner and one with the whole hand open but the index finger flexed away from the rest of the hand (Fig. 5). The latter hand position was what ended up getting used, as it allowed for greater extensibility into typing with more than one finger. The flatter hand is also more realistic, since this is how humans type.

C. Path Planning - Inverse Kinematic Controller

The next challenge was to implement an inverse kinematics based motion-planning approach for pressing the keys. The approach was to first determine the constraints of the hand trajectory. Within the x-y-z and rotational plane, there would be no freedom, as a key press requires a very accurate hit. Any inaccuracies will result in missing the desired key, or pressing adjacent keys. The orientation also had to be constrained fully because the extended finger must point down, or else the robot hand will not be able to press a key. The constraints and joint centering costs can be summarized as follows:

$$\begin{aligned} c &= ||q - q_{nom}||^2 \\ f_p(q)_x &= {}^W p^k{}_x \\ f_p(q)_y &= {}^W p^k{}_y \\ f_p(q)_z &= {}^W p^k{}_z \\ {}^W R^k &= f_R(q) \end{aligned} \tag{3}$$

Equation (3) describes the x,y,z, and rotational constraints and the joint centering cost, where $f_p(q)$ describes the forward dynamics function. To simulate the process of pressing a key, an inverse kinematic controller was developed to (1) hover over a key, (2) press a key. These final x-y-z positions are stored in a dictionary mapping, as mentioned in the section III-C. The hovering position is determined by adding a small offset to the z position of the key position to indicate a position above the key.

We predetermined which finger will press each key. The keyboard was divided into 3 parts for each of the non-thumb fingers to press. Each of the three sections are assigned to a finger to type (Fig. 6). In the code, we switched fingers by setting q_{nom} and the frame of the end effector. The initial q_{nom} for each finger was chosen by manually setting the joint positions using our study of the link correspondence. The frame of the end effector was chosen to be the link of the fingertip we were typing with. Using multiple fingers

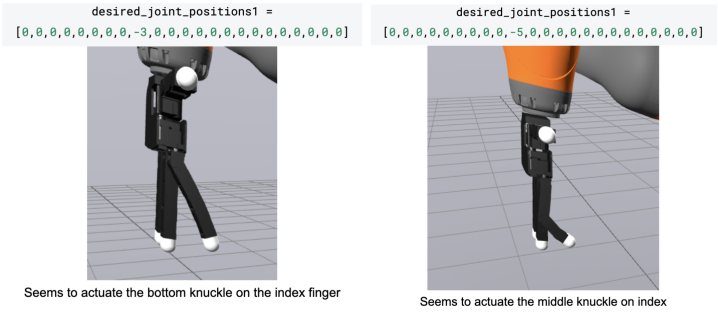

Fig. 4. Joint mapping exploration. By setting each of the 23 joints to a value, we could explore how each value corresponds to a joint.

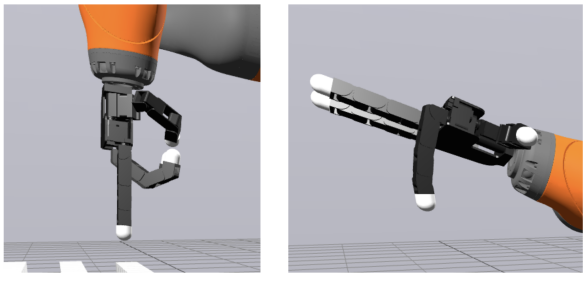

Fig. 5. Considered hand positions. The left example curls all fingers except the one typing. The right example keeps fingers extended, and flexes only the typing finger. The final robot uses the right example's hand position.

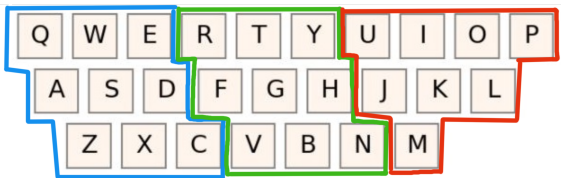

Fig. 6. Blue regions of the keyboard were programmed to be pressed by the index finger, green for the middle, and red for the pinky.

instead of one is more realistic. Additionally, humans have natural instinct for which finger to use for which key, so predetermining which finger presses which key is a reasonable solution.

V. Challenges

In the process of building our typing robot, there were three primary challenges: simulation lag, inaccuracy of key presses, and model convergence.

A. Simulation Lag

As more keys were added to the environment, the simulation became extremely laggy. This became an issue, as testing even a simple key press became an extraordinarily long process.

Before adding many keys, the simulation required relatively few calculations to run. However, with each key added, more complex spring and collision interactions were added as well, thus greatly increasing the amount of computation required. For each set of keys, extra rigid bodies were not needed, as the AddShape function does so internally. Removing the extra rigid bodies greatly improved the simulation speed.

As for the time step, it was initially much smaller, at 0.0001. However, as the simulation became more complex, performing such granular calculations overloaded the system. A time step of 0.01 was chosen, which is granular enough to accurately calculate behaviors, while also large enough to prevent lag.

B. Inaccurate Key Presses

When calculating the joint positions for pressing a key, the project was initially planned to calculate only the pressed position. However, this became an issue, as the the robot hand ended up hitting other keys on its trajectory, thus destroying the keyboard or typing inaccurately. The issue is illustrated in Fig. 7.

As a result of such collisions, an additional hand position was calculated above each of the keys in order to prevent the dangerous diagonal trajectories. This approach is was discussed in section IV-C. Because the robot takes some time to adjust to the correct joint position, we add an additional copy of the hovering pose before the robot presses down to allow the robot's momentum to stabilize.

C. Model Convergence

Sometimes, the model would be unable to calculate a joint position for keys farther away such as Q and P. The robot usually had no problems clicking keys close to the center, such as G or H. We solved this problem by reducing the

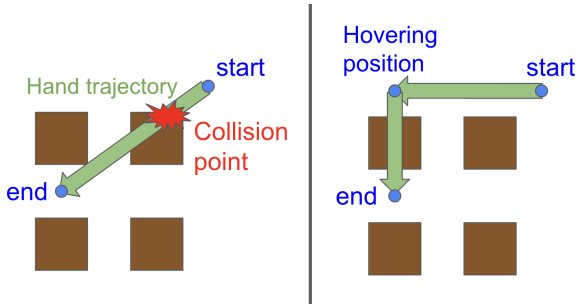

Fig. 7. Trajectory of collisions. Illustrated on the left, the red hand trajectory causes collisions with neighboring keys when taking on a diagonal path from above a key to a pressed position. Illustrated on the right is the solution: first reach a hovering position over the key, and then press down onto the key.

spacing between keys, to ensure that the robot would not have to stretch. Placing the keyboard closer to the robot, along with making the keyboard a bit higher up, both helped with model convergence.

In the future, we also would like to type the spacebar with the thumb rather than index finger, since we were unable to get the model to converge using the thumb.

VI. Results

The robot is able to type messages with high accuracy. The accuracy is evaluated by the following metrics:

- Ability to hit the desired key and have it register.
- Ability to hit the desired key without disturbing keys around it.

The robot is able to accomplish both 100% of the time. Below includes the tested messages and the reasoning:

- "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG": to ensure every key works.
- "PQWOLASKMZXN": to test the ability of the robot to type both far-apart letters and close-together letters.
- "GGGGG": to test the ability of the robot to type a single letter multiple times.
- "P H Q Z M ": to test the ability of the robot to type a space when coming from various regions of the keyboard.

.

The robot is able to accomplish all with 100% accuracy.

VII. Discussion

Our typing robot accomplishes the project's initial goals were: it is able to type multi-letter messages using different fingers. The successes and limitations of the system will be described in this section.

A. Successes

Our project demonstrates the feasibility of having a robot type, and also demonstrates an additional use case for a dexterous robot hands. Furthermore, the high accuracy of the system demonstrates the feasibility of its application in prosthetics, as the arm is capable of performing highly accurately, much like that of a real human arm.

B. Limitations

While the system is highly accurate, it is not adaptable or efficient. There are no optimization algorithms for determining the best finger to use. This means that the robot may type using a finger that is less optimal. For example, if the robot needs to type "QE", it will type "Q" with the index finger and then "E" with the index finger as well. However, a human will likely do this differently, as the path that requires the least amount of change would be typing "Q" with the index finger, and "E" with the middle finger. Next steps that could be taken would be to employ an optimization algorithm that can help determine the best finger to use in each situation, rather than predetermining which finger will type which letter. With this optimization, the robot will likely be able to type faster as well, as the each letter will require less translational movement time from the robot arm.

VIII. Member Contributions

Cynthia Zhang contributed to the process of figuring out how to move the robot arm and hand within a simulation, how to simulate a keyboard key, and how to set up motion planning with inverse kinematics for a single finger. Monica Chan contributed to building the keyboard up from a single key, detecting and displaying key presses, model convergence for the single finger setup, and extending inverse kinematics to multiple fingers across the keyboard.

References

[1] S. H. Yeon, "Playing Piano with a Robotic Hand," Massachusetts Institute of Technology, Cambridge, MA, 2021. [Online]. Available: https://github.com/seongho-yeon/playing-piano-with-a-robotic-hand/blob/main/report.pdf. [Accessed: Nov. 3, 2023].

[2] Y. -F. Li and C. -W. Huang, "Force control for the fingers of the piano playing robot — A gain switched approach," 2015 IEEE 11th International Conference on Power Electronics and Drive Systems, Sydney, NSW, Australia, 2015, pp. 265-270, doi: 10.1109/PEDS.2015.7203429.