- Technical Report -

Motion Planning of a Single-Motor Self-Adjusting Robotic Hand

Yoav Golan, Amir Shapiro
Department of Mechanical Engineering
Ben-Gurion University
Be'er Sheva, Israel
yoavgo@post.bgu.ac.il, ashapiro@bgu.ac.il

Elon Rimon
Department of Mechanical Engineering
Technion Institute of Technology
Haifa, Israel
rimon@technion.ac.il

Abstract—**Grasping objects with robotic grippers is still a difficult task. Grippers often sacrifice either flexibility or reliability to gain the other. This technical report introduces a robot hand design that attempts to balance reliability and flexibility with a novel concept. The hand uses the environment to adjust itself prior to grasping an object. This adjustment allows flexibility in the grippers application. The hand has only a single motor, that drives a deterministic actuation medium. This contributes to the hand's reliability. The hand's adjustment prior to grasping is the unique part of the design and operation of the gripper. In this work, the basic design principles of the hand are presented. The majority of the report, however, is focused on the motion planning principles behind the hand reconfiguration. Specifically, each finger in the hand can be changes in angle and fingertip distance form the hand's center. This allows full-freedom of the fingertip placement, but poses a challenge in reconfiguration. The tasks of angle adjustment and fingertip distance adjustment are both elaborated on in this report, providing analytic, time efficient solutions for both and their combination.**

I. Introduction

Robotic grippers are practical only when they are extremely reliable. Grippers that can manipulate a wide variety of objects generally have to sacrifice reliability to gain flexibility. Conversely, reliable grippers tend to be very limited in the objects they can grasp. The quintessential gripper would be able to grasp a wide variety of objects reliably. Towards this end, we introduce a single-actuator robot hand that can be adapted to grasp a wide variety of objects. The gripper is deterministic (not soft), allowing predictability in its operation. This predictability increases grasp reliability. The hand can be adjusted to match an object's shape prior to grasping. Simplicity is maintained by having only a single actuator. The adjustment flexibility is grated by a novel concept of using the environment. By performing a series of actions with the hand against the environment, the hand's configuration can be changed sequentially. In other words, by investing some time in configuring the hand prior to grasping, a tailored, reliable gripping device is synthesized for a specific object. This planar hand is an important step towards a combination of flexibility and reliability in robotic grippers.

Modern robot hand research focuses on the generalization of gripping. Some researchers have focused on "classic" grippers and sophisticated means of decision making to securely grasp items. For instance, Mahler *et al.* [1] use a vacuum gripper or parallel-jaw gripper to perform bin picking tasks. Machine learning is used to generalize object grasps from a training set, allowing the system to select the correct gripper, item, and approach point to perform a picking operation. Other researchers have elected to develop underactuated hands that can adapt to an object, such as Backus [2] and Dollar [3]. The emerging field of soft robotics [4] has produced many interesting grippers designed to grip a variety of objects, e.g. [5], [6], [7], [8]. While underactuated and soft robotics are making significant progress towards industrial use, they still cannot compare with the reliability and robustness of classic industrial grippers.

This report describes a planar, minimalistic planar-acting robot hand that uses the *environment* to change its configuration. The hand uses a single degree of freedom (DOF), driven by a single actuator. The hand is not soft, underactuated or shape conforming. Rather, it is rigid and predictable in its operation, similar to classical industrial grippers. Adaptability is achieved by reshaping the hand's configuration before grasping. In other words, the hand is tailored to the object it is meant to grasp, before attempting to grasp the object. In contrast to specialized hands designed to grasp specific objects, there is no need to redesign and manufacture a new hand whenever a new object is given. Instead, our hand can be adjusted using the environment prior to its use.

The concept described in this report works as follows. The robot arm observes an object and determines the desired grasp, without being constrained by the hand configuration. The robot arm then performs a series of adjustments to the hand configuration, to suit it to the best grasp of the given object. This is the exact opposite of works like Mahler's [1] that dedicate their effort to selecting the best possible grasp of an object using the available fixed-structure grippers. The main caveat of our approach is the time needed to perform the physical alteration of the hand prior to grasping. However, this extra reconfiguration time is only needed when grasping new disparate objects, and is not necessary when grasping multiple identical items sequentially. Therefore, we expect that such a hand could save time and costs in robotic tasks where a gripper must grasp multiple items of one kind, and later grasp multiple items of another kind. Furthermore, when two different items are to be grasped with enough time

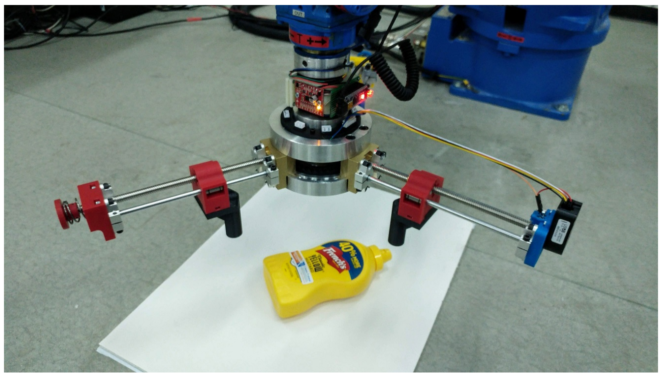
Fig. 1. A picture of the hand prior to grasping an object.

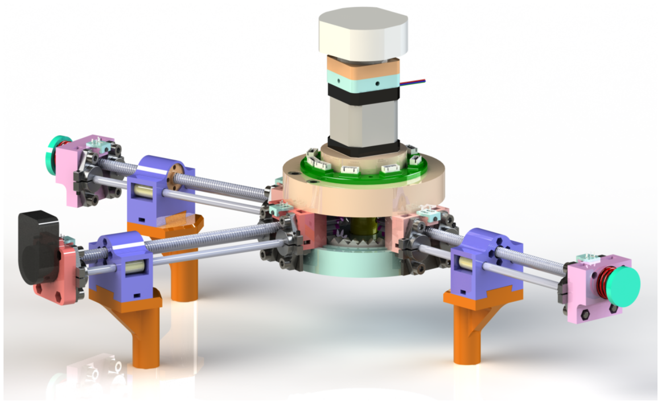
Fig. 2. A rendering of the hand with two fingers and a thumb.

between them to re-arrange the hand, our system will improve the robot hand's flexibility without wasting system operation time.

This technical report will detail the basic mechanical principle of the robot hand in Section II. In Section III the method for selecting the grasp configurations is detailed. The main focus of this report, however, is in the motion planning behind the hand's reconfiguration. When changing from one configuration to another, the hand changes both the relative angle, and distance of each fingertip. This overall change is performed by sequential steps, that are non-trivial to obtain. Therefore, Section IV focuses on the synthesis of the *adjustment procedure*. Section V shows experiments performed with a real hand and robot system, and Section **??** concludes the report.

II. Design and Method of Operation

In this section we present the design of a variable configuration robotic hand. The novel mechanisms used in the hand allow is to be deterministic, while maintaining flexibility. This is done through adjustments made using the environment.

A. Design Overview

The robotic hand is comprised of a single motor, a central body, and digits. One of the digits is unique, and is termed the "thumb", but only as a titular distinction from the other digits– the "fingers". There are between 0-7 fingers in addition to the thumb, for a total of 1-8 digits in the hand. Adding or removing a finger requires partial disassembly of the hand; therefore, changing the number of fingers is normally a pre-determined choice, and not part of normal operation. The number of digits is selected according to the expected type of grasping tasks. Most single actuator grippers use two to four digits, therefore most of our analyses, figures and demonstrations use these numbers of digits.

The central body acts as a hub, connecting the digits and the motor. Two opposing face gears are centered in the body, and are powered by the motor. Each digit interacts with the gear system on the central body, transferring power from the motor to the digits. In addition, the central body acts as a circular rail system, allowing the fingers to move angularly about the central axis, providing that other conditions allow the movement. The thumb is rigidly attached to the central body, so that it cannot move angularly about the central axis. This is one of the unique attributes of the thumb, relative to the fingers. Each digit assumes a $45°$ wedge within the circular central body, therefore at most eight digits can be used, and the minimal angle between any two digits is $45°$. The more digits are used, the less angular freedom the system has, to a point where eight digits allows no angular movement of any digit.

Each finger is a detachable module that contains several parts. A specialized spur gear is mounted on a lead screw, and is designed to mesh with the face gears of the central body. A knob and spring are mounted on the opposite side of the lead spring, and are used to apply force to it from the environment. A *fingertip* is mounted on the lead screw and two linear rails, so that when the lead screw turns, the fingertip moves radially (towards or away from the center of the hand). At either end of the finger module is a block that houses the linear rails and the lead screw. The distal block houses the spring, and the proximal block houses the gear. The proximal block also has two male circular rail inserts, that interact with the female rails on the central body. Each block also has a simple button that is pressed when the fingertip reaches either extrema of its motion. When the knob is pressed, the lead screw, fingertip and gear all displace radially, relative to the blocks and linear rails. A rendering of the finger module is depicted in Fig. 3.

The thumb module is similar to the finger module, with a few exceptions. Instead of a knob and spring, an optical encoder is mounted on the lead screw at the distal end of the thumb. The encoder measures the rotation of the lead screw. Additionally, the proximal block is bolted to the central body, to rigidly attach the two. Unlike the finger, the lead screw, fingertip and gear cannot be displaced radially.

The robotic hand attaches to a robotic arm at the central body, or by the motor. In our implementation, a stepper motor us used, and the robotic arm is directly attached to the motor. An object is intended to be grasped by the fingertips. These fingertips can be of any type; in our implementation simply cylinders are used. However, more complex fingertips, such as dual-friction fingertips, could be used without issue [9].

B. Mechanical Principle

The motor directly powers the upper face gear. The thumb spur gear is permanently meshed with both face gears, and therefore the bottom face gear always rotates in the opposite direction of the top face gear, at the same angular velocity. When a finger spur gear is meshed with the face gears, a transfer of power is enabled. If the motor rotates, the face gears rotate in turn. The face gears turn the finger spur gear, turning the finger lead screw, and moving the fingertip radially, towards or away from the hand's center, depending of the motor's rotation. In this way, turning the motor leads to the radial change of all the fingertips meshed with the gears. The gears of all the digits are identical, therefore a single turn of the motor will result in an identical linear movement of each of the fingertips, relative to the hand's center.

The thumb is not free to rotate about the central body because it is rigidly attached to it. When a finger's gear is meshed with the face gears, the finger is not free to rotate about the central body, because the face gears will prohibit the motion. Since the gear ratio for the face gears is always 1:-1, a spur gear cannot force angular motion within the the gear system. In other words, while a finger is meshed with the face gears, it cannot move angularly. When the face gears rotate, they keep the finger module in place angularly, while moving the fingertip radially.

The mechanical principle that allows adjustment of the hand is as follows. When the knob of a finger is pressed against the spring, the lead screw, fingertip and gear are all displaced radially. This displacement is not large, but it is enough to *disengage* the spur gear from the face gears. When a finger is disengaged, two adjustment possibilities emerge. Firstly, when the spur gear is disengaged, and the motor turns– the fingertip does not move radially. More importantly, the fingertips of the other digits continue to move undisturbed. This means that when a finger is disengaged, rotation of the motor results in a change in the relative distance between the disengaged fingertip and the other fingertips. Secondly, when the finger is disengaged, it can be rotated about the central body freely (the meshing of the gears previously prevented the rotation). This means that when a finger is disengaged, the angle of the finger can be changed relative to the other digits.

Once the pressure on the knob is released, the spring forces the spur gear to mesh back with the face gears. Once re-engaged, the finger can no longer rotate freely, and the fingertip will resume its radial motion when the motor turns. A series of disengagements, motor turns, finger rotations and re-engagements can change the angles between the fingertips and the relative distances between them, within the physical limitations of the hand. Note that the application of force on a fingertip by a grasped object does not disengage the finger, because the disengagement happens in the opposite direction. After each adjustment, the hand is essentially a rigid robotic hand with a single closing parameter, easily controlled and operated.

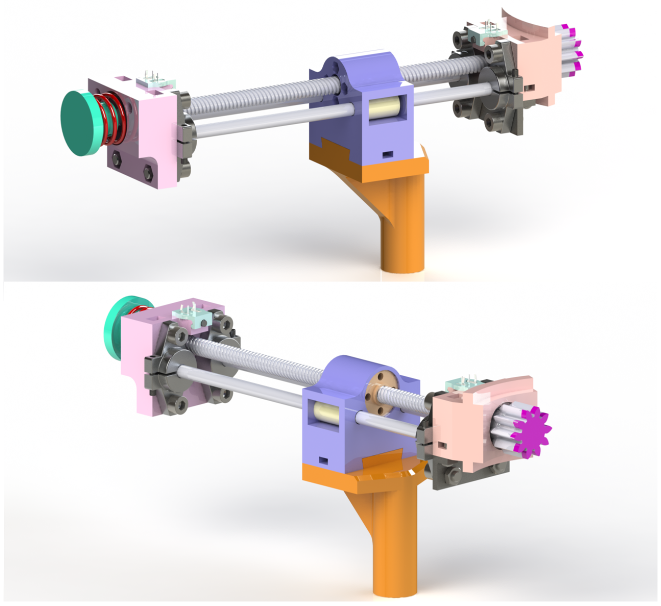
Fig. 3. A CAD rendering of the Finger module, with two points of view.

III. Selecting the Best Grasp

This section details a novel approach of selecting a grasp configuration for an object. When using a non-configurable robot hand, any grasp configuration must conform to the hand shape. For instance, if an equidistant, three-fingered parallel-jaw hand (e.g. [10]) is used, only grasp configurations that constitute equilateral triangles may be considered. Our approach utilizes the reconfiguration ability of the hand to maximize grasp quality by relaxing the finger positioning constraints. A grasp configuration is defined as the fingertip placements on the object's perimeter. A *hand configuration* is defined as a combination of the grasp configuration and the location of the hand's center. Although our hand allows total freedom in grasp configuration selection, physical constraints still exist, so not every hand configuration is possible. Firstly though, let us examine the decision process for finger placement– the grasp configuration.

The object is given as a polygon in configuration space. I.e, the object has been represented as a polygon, and dilated by the radius of the fingertips. This means that any point on the boundary of the configuration-space object corresponds to a fingertip-object contact point. The friction coefficient between the fingertips and the object μ is known, as is the number of fingers n. We now proceed to choose the grasp configuration. To do this, we use Monte-Carlo simulation of grasp configurations. We discretize the polygon perimeter as a set of points, each with its own location and surface normal direction. We then randomly place n fingertips at different points on the polygon perimeter, and test the grasp quality. There are many grasp quality measures, and it is not this paper's intention to advocate one or the other. Our procedure grants the user the option to choose between *wrench space sphere radius* and *grasp matrix ellipsoid* quality measures, although any other quality measure can be used. These

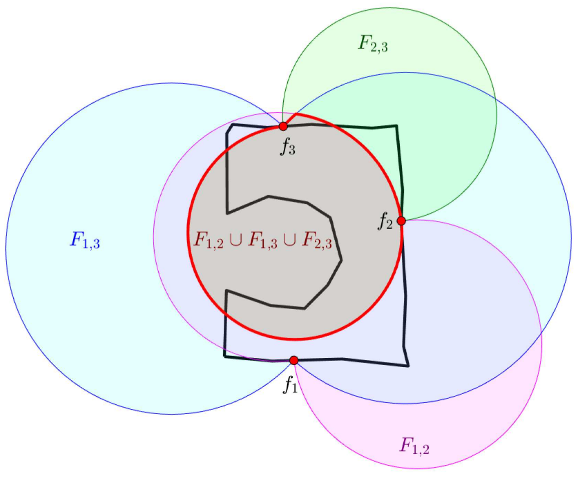

Fig. 4. An example of the angular geometric constraints of finger placement. The object (black outline) is grasped by three fingertips (red dots). The angle between two digits must be at least $\gamma = 45°$. For each pair of fingers, the union of two circles bounds the permitted area. In this instance, fingers 1 and 2 only allow the placement of the hand center within the double-circle magenta shape $F_{1,2}$. The intersection of all the pairs results in the allowed area (red border).

quality measures and others can be found in [11, pp. 321-348].

The grasp is evaluated, and its quality is marked according to the guidelines of the quality measure. If viable, the grasp configuration enters a list of possible grasp configurations. After exhausting the user-defined number of grasp attempts, the list is sorted by grasp quality. We now have a list of n feasible grasp configurations, sorted by their quality. Not all of these grasp configurations are actually possible, since the robotic hand we use has physical limitations. A hand configuration is possible if a possible hand center exists, given the grasp configuration. Therefore, we can examine each grasp configuration to analyze its potential as a hand configuration.

Starting from the best grasp configuration in the list, we test to see if a hand configuration can be synthesized. I.e, we test to see if a point P exists that is the center of a hand with its fingers at the grasp configuration. This test is performed by converting the physical hand limitations to geometric constraints. Consider the point P representing the center of the hand. P must be within a circle of L_{max} radius from each fingertip placement, where L_{max} is the maximal extension of a fingertip. This means that the center must lie inside the intersection of n circles centered at the finger placement points, each with radius L_{max}. Mathematically put, the center must be within the area $\mathcal{A}_1$:

$$\mathcal{A}_1 = \bigcap_{i=1}^{k} \mathcal{D}_i \tag{1}$$

where $\mathcal{D}_i$ is a circle of radius L_{max} centered at the ith fingertip placement. This can be seen in Fig. 5, as magenta circles surrounding the fingertip placements.

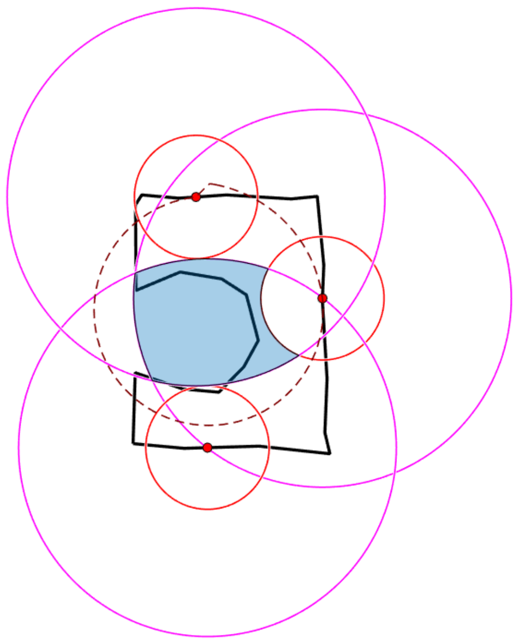

Fig. 5. An continuation of the example from Fig. 4. We add geometric constraints pertaining to the distance of fingertips from the hand center. The object (black outline) is grasped by three fingertips (red dots). The hand center must lie inside L_{max} from each fingertip (magenta circles). The hand center cannot lie within L_{min} from any fingertip (red circles). To conform with angle restrictions, the hand center must lie inside the area depicted in Fig. 4, seen here as a dashed line. The blue area is the result of these restrictions, where a hand center can be placed.

Similarly, the hand center cannot lie too near to a fingertip placement, since there is a minimum extension L_{min}. Therefore the center *cannot* lie within $\mathcal{A}_2$, defined as:

$$\mathcal{A}_2 = \bigcup_{i=1}^{k} \mathcal{E}_i \tag{2}$$

where $\mathcal{E}_i$ is a circle of radius L_{min} centered at the ith fingertip placement. This can be seen in Fig. 5, as red circles surrounding the fingertip placements.

The final physical limitation is the angular one. Two neighboring digits cannot be at angles less than γ apart. This limitation can be described geometrically as circles as well. Let us take two fingertip placements, f_i and f_j . The hand center P must be located such that the digit vectors are no more than γ degrees apart, or:

$$\angle(\overrightarrow{f_i - P}, \overrightarrow{f_j - P}) \geq \gamma. \tag{3}$$

The set of points P that conform to this rule lie within either of two overlapping circles. Each circle is bounded by f_i and f_j. Each point on the perimeter of either circle is such that $\angle(\overrightarrow{f_i - P}, \overrightarrow{f_j - P}) = \gamma$. If the distance between f_i and f_j is $d_{i,j}$, than the radii of the two circles are:

$$r_{i,j} = \frac{d_{i,j}}{2\sin(\gamma)}. \tag{4}$$

The center of the hand, therefore, must lie within the area:

$$\mathcal{F}_{i,j} = \mathcal{F}^1_{i,j} \cup \mathcal{F}^2_{i,j} \tag{5}$$

where $\mathcal{F}^1_{i,j}$ and $\mathcal{F}^2_{i,j}$ are the two circles with radius $r_{i,j}$ that have f_i and f_j lying on their perimeters. The hand's center must be inside this area for every pair of fingers f_i,f_j. Therefore, the hand's center must lie within:

$$\mathcal{A}_3 = \bigcap \mathcal{F}_{i,j} \;\text{ for } 1 \leq i,j \leq n,\; i \neq j\ . \tag{6}$$

This constraint can be seen in Fig. 4 for a three-finger hand, as three unions of circle pairs.

Finally, we combine the geometric constraints to obtain the valid area. Any point in this area is a physically feasible placement for the hand center:

$$\mathcal{A} = \mathcal{A}_1 \cap \bar{\mathcal{A}}_2 \cap \mathcal{A}_3\ . \tag{7}$$

This area is illustrated in the example in Fig, 5.

Any point in $\mathcal{A}$ that allows a penetrating grasp is valid, and as far as grasp quality they are identical. However, some hand center positions are better in other regards. We can identify *special centers* that have certain advantages, and try them before moving on to other considerations. For instance, a special center for three-fingered hands is the center of the circle defined by the three fingertip placements. This center has two advantages: 1) Each of the three fingertips is equidistant from the center. If the previous grasp configuration was also equidistant, the distance adjustment procedure is *exceedingly short.* 2) If we define a triangle by the three fingertip placements, we note that extending or retracting the fingertips creates a similar triangle. Similar triangles can be used to define caging regions on polygons [12], potentially increasing grasp reliability and robustness.

If the preferred special center is not viable, we randomly sample several hand center options inside the allowed area $\mathcal{A}$. Each of these options is viable, but we prefer hand centers that require shorter reconfiguration procedures of the hand. Each hand center P_i corresponds with a hand configuration $C_i = (\vec{\theta_i}, \vec{d_i})$. Starting from the hand's current configuration C_0, each of the configurations C_i may take a different number of adjustments to achieve. Therefore, we construct adjustment procedures for every configuration, as mentioned in the following Section **??**. After constructing the adjustment procedure for each configuration, we choose the hand center that requires the shortest procedure. A distance adjustment protocol's length is determined by the number of adjustments, and not their retracting or extending values. This is because switching from pressing one finger or another is a more time expensive task than simply retracting or extending the fingers. At this point, a number of valid hand configurations have been found. In the next section, we detail the method of constructing and executing the adjustment procedure, which allows us to select and utilize one of these hand configurations.

IV. Adjustment Procedure

This section describes the methods proposed to adjust the hand to a desired configuration. Given the initial configuration, and desired configuration of the hand ,a series of adjustments need to be carried out. These adjustments are easily divisible into two problems– the adjustment of digit orientation, and the adjustment of digit extension. Each of these problems is addressed and solved separately, and the integration of the two is discussed.

For convenience, we will explain the adjustments methods in terms of *planning*. I.e., we describe the adjustments as a series of actions, each of which has a set of prerequisites and results, changing the state of the system. The formulation of the problem as a planning problem is helpful to gain intuition, however we do not use planning algorithms to solve the problem. We use an analytic approach to reach the goal state from the initial state, resulting in a time-efficient solution.

Assume a hand with n digits, $f_0, \ldots, f_{n-1}$, where f_0 is the thumb, and the digit numbers are arranged counter-clockwise starting from the thumb (top view). Each digit's fingertip has two parameters that define its configuration– d_i and θ_i. d_i is the distance, or *extensions*, between the fingertip and the center of the hand, and θ_i is the angle between the digit and the hand. Since the thumb is rigidly attached to the hand, we assign $\theta_0 = 0°$ for convenience. Thus, the configuration of the hand can be defined as:

$$\vec{Q} = \left(\vec{\theta}, \vec{d}\right) = (0, \theta_1, \ldots, \theta_{n-1}, d_0, d_1, \ldots, d_{n-1}) \tag{8}$$

The problem is formulated as follows: we wish to find a series of adjustments that will change the congiguration from a starting configuration $\vec{Q}^S$ to a target configuration $\vec{Q}^T$. Assuming there are j steps, a valid plan P will be of the form:

$$P = \vec{Q}^S \rightarrow \vec{Q}^1 \rightarrow \vec{Q}^2 \rightarrow \cdots \rightarrow \vec{Q}^{j-1} \rightarrow \vec{Q}^T \tag{9}$$

Each step of the plan involves an action of some sort. The possible actions are: a) rotating the motor in either direction by some amount, b) disengaging a finger, c) rotating a finger about the hand, d) engaging a finger. Each adjustment has perquisites; for instance, adjustments c) and d) require a finger to be disengaged. Note that changing the orientations $\vec{\theta}$ has no effect on the extensions $\vec{d}$, and vice versa. Therefore, we can solve each of the two problems separately without affecting the other. Firstly, we will find a plan to solve the orientations:

$$\vec{\theta}^{\,S} \rightarrow \vec{\theta}^{\,1} \rightarrow \cdots \rightarrow \vec{\theta}^{\,T} \tag{10}$$

Then, we will find a plan to solve the extensions:

$$\vec{d}^{\,S} \rightarrow \vec{d}^{\,1} \rightarrow \cdots \rightarrow \vec{d}^{\,T} \tag{11}$$

Lastly, we will show how the two can be integrated to form a complete plan (Plan (9)).

A. Adjustment of Digit Orientation

In order to change the orientation of a single finger $\theta_i^k \rightarrow \theta_i^{k+1}$, the following actions must be taken. Firstly, the i^{th} finger must be disengaged. Next, the finger must be rotated about the hand by some angle β, changing the orientation. Lastly, the i^{th} finger must be re-engaged. These three actions, in this order, result in a change of orientation of the i^{th} finger by β, so that $\theta_i^{k+1} = \theta_i^k + \beta$. Note that the fingertips are all identical, therefore the target configuration can always be re-ordered by ascending values: $\theta_i^T < \theta_{i+1}^T$.

Given that each of these three actions is necessary and irreplaceable, we will group the three actions as a single action called a "rotation step". A rotation step can only be performed if there is no other digit that would physically collide with the rotated finger. For instance, let us assume there are three digits at a starting orientation $\vec{\theta}^{\,S} = (0°, 60°, 110°)$ and a target orientation $\vec{\theta}^{\,T} = (0°, 180°, 270°)$. Finger f_1 cannot move from its starting position of 60° to its target position 180○, because it would have to pass through f_2 at its starting position 110°). However, f_2 can be moved from its starting position 110°) to its target position 270°) without collision.

The absolute lowest number of rotation steps necessary to achieve the target orientation configuration, assuming ${\theta_i}^S \neq {\theta_i}^T,\ i \geq 1$, is $n-1$, where n is the number of digits, and $n-1$ is the number of fingers. In other words, provided that no finger starts at its target position, each finger needs to be rotated at least once. We will now show that, in fact, each finger needs to be rotated *exactly once*.

Theorem 1. *Given a start and target orientation configuration, each finger of the hand needs to be rotated at most one time.*

Proof. Let us assume a hand has n fingers at an initial orientation $\vec{\theta}^{\,S} = \left(0, \theta_1^S, \ldots, \theta_n^S\right)$, and a target orientation $\vec{\theta}^{\,T} = \left(0, \theta_1^T, \ldots, \theta_n^T\right)$. Starting from n, we will examine each finger to check if it can be re-oriented to its target position without interruption. Let us assume that f_n cannot be re-orientated due to collision. By definition, this means that $\theta_{n-1}^S + 45° > \theta_n^T$. Let us move on to f_{n-1}, and again assume that it cannot be re-oriented directly to its target position. Similarly, this means: $\theta_{n-2}^S + 45° > \theta_{n-1}^T$. Continuing this way, assuming each finger cannot be directly re-oriented, we eventually reach f_1. the only way f_1 cannot be directly be re-oriented is if $\theta_0^S + 45° > \theta_1^T$. Since $\theta_0 = 0$ by definition, the collision condition can be re-written as $45° > \theta_1^T$. A valid target position for the first finger is $\theta_1^T \geq 45°$, therefore the collision condition cannot be met and f_1 can be directly oriented to its target position. □

B. Adjustment of Digit Extension

Unlike the adjustment of the digit orientation, the adjustment of digit extension does not have a simple, straightforward solution. To change the extension of a single finger relative to the other digits, one must first disengage he finger, rotate the motor, and then re-engage the finger. This will cause every digit to extend/retract except the finger that was disengaged. This group of actions effectively changes the relative distance of the disengaged finger, but practically changes the extension of every other digit to do so. This makes it difficult to intuitively select a series of actions that will end in the target configuration $\vec{d}^{\,T}$. For example, let us examine a four-digit hand at the 17^{th} step of it's plan. The current extension vector is $\vec{d}^{\,17} = \left(d_0^{17}, d_1^{17}, d_2^{17}, d_3^{17}\right)$. If we disengage f_2, rotate the motor so that the other fingertips extend be δ, and re-engage f_2, the resulting extension vector will be $\vec{d}^{\,18} = \left(d_0^{17} + \delta, d_1^{17} + \delta, d_2^{17}, d_3^{17} + \delta\right)$.

Let us define two action sequences and their results. We will hereby refer to the these sequences as "moves". The first move is a rotation of the motor when no fingers are disengaged. This results in an extension of all the digits by an equal amount: $\vec{d}^{\,k+1} = \vec{d}^{\,k} + \delta$. The second move is the following action sequence: disengagement of the jth finger f_j, rotation of the motor, and re-engagement of f_j. This results in an extension of all the digits by δ except for f_j.

Adjusting the extensions of the fingers in this way would not be difficult if there were no physical limitations to the system. However, a fingertip cannot be extended to any arbitrary distance, and there are physical limitations of the fingertip movement in the fingers: $d_i \in \left(d_{min,f}, d_{max,f}\right), i \geq$. The thumb has a similar limitation with slightly different extrema: $d_0 \in (d_{min,t}, d_{max,t})$. It is fairly easy to see that a target extension may not be achievable within a single "move". As an example, let us assume a three digit hand has an initial extension vector $\vec{d}^{\,S} = (80, 160, 120)$ and a goal extension vector $\vec{d}^{\,T} = (190, 40, 150)$. Let us further assume the physical limitations $d_{min,f} = d_{min,t} = 0$, and $d_{max,f} = d_{max,t} = 200$. There are three possible moves. Firstly, we could rotate the motor without any disengagement: $\vec{d}^{\,S} = \vec{d}^{\,S} \pm (\delta, \delta, \delta)$. Secondly, we could disengage f_1 and rotate the motor. This would result in: $\vec{d}^{\,S} = \vec{d}^{\,S} \pm (\delta, 0, \delta)$. Thirdly, we could disengage f_2 and rotate the motor. This would result in: $\vec{d}^{\,S} = \vec{d}^{\,S} \pm (\delta, \delta, 0)$. It is clear that with of these moves, there is no δ that would result in the goal extension of a digit without exceeding the limits of another. Furthermore, there is no real benefit in a single digit reaching its goal extension, given that the following move will immediately displace it.

To understand the analytic solution, let us start with a simple two digit hand. The hand has a starting extension vector $\vec{d}^{\,S} = (50, 150)$ and a target extension vector $\vec{d}^{\,T} = (80, 60)$. The physical limits of the digits are: $d_0 \in (0, 200),\ d_1 \in (20, 180)$. We can plot the start and target extensions as points in a 2D configuration space, as shown in Fig. 6. The configuration space is bounded by a rectangle that represents the physical limits of the digits. The basis movement vectors are $m_1 = \langle 1, 1 \rangle, m_2 = \langle 1, 0 \rangle$. The first is achieved by rotating the motor without disengagement, and the second is achieved by disengaging f_1 and rotating the motor. We plot the movement vectors in the 2D configuration space to indicate the space of possible configurations that can be achieved in a single move.

It is extremely unlikely that one of the movement vectors will intersect the target point. Therefore, uninformed placement of the next configuration step is ill-advised. Instead, realize that one move before the target is reached, the configuration will lie on one of the movement vectors extending from $\vec{d}^{\,T}$. Therefore, we extend the two movement vectors from the target, knowing that any configuration that lies on these lines will be one move away from reaching the target configuration. While it may be exceedingly unlikely for a line to intersect a point in 2D space, it is guaranteed that two lines will intersect in 2D space, unless parallel. Therefore, we search for the intersection between the movement vectors emanating from the start configuration and the movement vectors emanating from the target configuration. If there is an intersection within the physical constraint box, a solution is found. Observe Fig. 6. two movement vectors emanate from the starting configuration $\vec{d}^{\,S} = (50, 150)$, and two movement vectors emanate from the target configuration $\vec{d}^{\,T} = (80, 60)$. These four lines have two intersection points, at $(170, 150)$ and $(-40, 60)$. The second intersection is outside of the physical limits, marked by a black border in Fig. 6, therefore is not a valid move. We can now see a valid path from start to target, marked by a blue line in Fig. 6. The path from start to target is: $(50, 150) \rightarrow (170, 150) \rightarrow (80, 60)$. Physically, this path requires two moves, and the following series of actions: Starting at the target configuration, disengage f_1, and rotate the motor to cause an extension of $\delta_1 = 120$ mm. This first move will result in a new configuration $\vec{d}^{\,1} = (50, 150) + (120, 0) = (170, 150)$. Next, re-engage f_1, and rotate the motor to cause an extension of $\delta_2 = -90$ mm. This move will result in a new configuration $\vec{d}^{\,2} = (170, 150) + (-90, -90) = (80, 60) = \vec{d}^{\,T}$. Thus, we have found a valid path between the start configuration and target configuration, with two steps.

Next, let us observe a more difficult 2D problem, where the starting configuration is still $\vec{d}^{\,S} = (50, 150)$, but the target is $\vec{d}^{\,T} = (190, 40)$. If we follow the same procedure as before, we will quickly find that the movement lines emanating from the start and target configurations intersect at two points that are both outside of the physical boundaries. Therefore, there is no two-step solution for our problem. To solve this, we add several intermediate configurations to our configuration space. Specifically, we add the intersections between the movement vectors emanating from the starting configuration, and the physical bounding box. These intersections add four configurations to explore, $(80, 180)$, $(0, 100)$, $(0, 150)$, $(200, 150)$. Each of these configurations is termed a *node*. Each node can be *expanded*, by emanating movement lines from its configuration. When expanding a node, we first check if there exists a valid intersection with the movement vectors emanating from the target (a valid intersection is within the physical boundaries). If so, there is no need to continue the expanse, because a valid path has been found. Otherwise, the expanse creates new nodes at the intersections between the movement vectors and the physical boundaries.

Time is the most important factor in our search for a path. Rotating the motor to change the extension values is much faster than disengaging, and then re-engaging a finger. Therefore, the fastest path will typically be the one with the fewest number of nodes, and not the path with the shortest euclidean distance. The number of nodes represent the number of disengagements and re-engagements that will need to be performed, while the euclidean distance roughly represents the number of motor turns. More precisely, the total distance traveled in the f_0 direction represents the number of motor turns, multiplied by the gear ratio. Understanding this, it follows that the fastest path will be found by performing a Breadth First Search (BFS) in our configuration space, with the first node being the starting configuration. The target is not well defined, because it can be any point on the movement vectors emanating from the target, but it is easy to determine when such a point is found, ending the search. This breadth-first search ensures that every two-step path possibility will be exhausted before searching for a three-step path, every three-step path will be exhausted before searching for a four-step path, and so on. In our second example depicted in Fig. 7, A three-step path has been found: $(50, 150) \rightarrow (200, 150) \rightarrow (90, 40) \rightarrow (190, 40)$. Note that there may be more than one valid path with the same number of steps. In this case, it is slightly beneficial to select the path with the least euclidean distance traveled in the f_0 direction. In most cases, however, there will be no significant time difference between paths, and so selecting the first found path arbitrarily is a reasonable choice.

Let us now extend this procedure to three digits. Now, we have a three dimensional configuration space, with three movement vectors: $\vec{m}_1 = \langle 1, 1, 1\rangle, \vec{m}_2 = \langle 1, 1, 0\rangle, \vec{m}_3 = \langle 1, 0, 1\rangle$. The physical boundaries are now represented as a cuboid. Again, we sill begin by drawing the movement vectors emanating from the target configuration. Unfortunately, just as it was extremely unlikely to intersect a line with a point in 2D space, it is extremely unlikely to intersect a line with a line in 3D space. This means that we cannot simply expand nodes in the same way as before, hoping to intersect the movement vectors emanating from the target. We do, however, know that if a path does exist, its last step must lie on one of the three movement vectors emanating from the target. Furthermore, to reach a movement vector, the second-to-last configuration must lie in a plane that coincides with a movement vector emanating from the target, and a different movement vector. This means that we can extend a 2D plane in 3D space, describing all configurations that are two steps away from the target, disregarding physical limitations. For instance, take the movement vector $\vec{m}_3 = \langle 1, 0, 1\rangle$, emanating from the target configuration $\vec{d}^{\,T} = (d_0^T, d_1^T, d_2^T)$. Let us now span two planes. We find the first plane's normal by vector multiplication of $\vec{m}_3$ and the first remaining movement vector $\vec{m}_1$:

$$\vec{n}_1 = \vec{m}_3 \times \vec{m}_1 = \langle 1, 0, 1\rangle \times \langle 1, 1, 1\rangle = \langle -1, 0, 1\rangle \tag{12}$$

we use the target configuration as a reference point, giving us a fully defined 2D plane P_1:

$$0 = \overrightarrow{P_1 d^T} \cdot \langle -1, 0, 1\rangle \tag{13}$$

Similarly, we can define the second plane:

$$0 = \overrightarrow{P_2 d^T} \cdot \vec{m}_3 \times \vec{m}_2 = 0 = \overrightarrow{P_2 d^T} \cdot \langle -1, 1, 1 \rangle \qquad (14)$$

While it may be extremely unlikely for a 1D line to intersect another 1D line in 3D space, a 1D line is guaranteed to intersect a 2D plane (unless parallel). Our path search algorithm is now reformed accordingly. First, we emanate three movement vectors from the target configuration. Next, we emanate two planes from each of the three aforementioned movement vectors. These six planes represent the locations from which the system is two moves away from the target configuration. A search is now initiated by expanding the start configuration node. Just as in the 2D case, if a movement vector emanating from the node intersects a physical constraint (in this case, any of the six 2D planes bounding the cuboid), it creates a new node. If a movement vector intersects one of the six planes within the cuboid, the search is paused in order to test the possibility of a path. Starting from the line-plane intersection point, a movement vector is extended within the plane, and the intersection point between this new movement vector and the movement vector emanating from the target is computed. If the point is within the cuboid, the search is terminated, because a valid path is found. If the line-line intersection point is outside the cuboid, the path is discarded and the search continues. Fig. 8 depicts the path found using this method, starting from $\vec{d}^{\,S} = (139, 71.5, 172)$ and ending at the target $\vec{d}^{\,T} = (6, 90, 81)$. The path has four steps: $(139, 71.5, 172) \rightarrow (0, 71.5, 33) \rightarrow (60.5, 132, 93.5) \rightarrow (48, 132, 81) \rightarrow (6, 90, 81)$

In the 3D case, we searched for the intersection of a 1D line with a 2D plane, which is relatively easy to find. Then, we searched for the intersection of a 1D line with a 1D line in the 2D plane, which is also easy to find. Lastly, we searched for the intersection betwen a 1D line with a 0D point, along a 1D line. It becomes obvious that the search is always performed between a 1D line and a $N-1$ hyperplane in $N-$D space. We can thus expand our method to any number of digits. To do this, we must make several generalizations. Assuming there are n digits in our hand, the physical boundaries of the digits can be represented in nD configuration space as an nD hypercuboid. The hypercuboid has $2n$ facets, each of which can be represented as a $n-1$ hyperplane. In 3D space we are accustomed to use the cross product to find normal vectors, however in higher or lower dimensions we are compelled to use a more general approach. Specifically, the $n-1$ hyperplane defined as normal to two $n-2$ hyperplanes is the null space of the two $n-2$ hyperplanes. We always start with the target configuration. We then extend movement vectors, planes etc. according to the dimension. For n dimensions, we extend the target $n-1$ times. For example, for $n = 5$ we take the 0D target and emanate 1D movement vectors from it. We then emanate 2D planes from the 1D vectors. Next, we emanate 3D spaces from the 2D planes, and finally emanate 4D hyperspaces from the 3D spaces. in 5D space, it is guaranteed that a 1D vector will intersect a 4D hyperspace (unless parallel). The search procedure is then exacted, as always using the

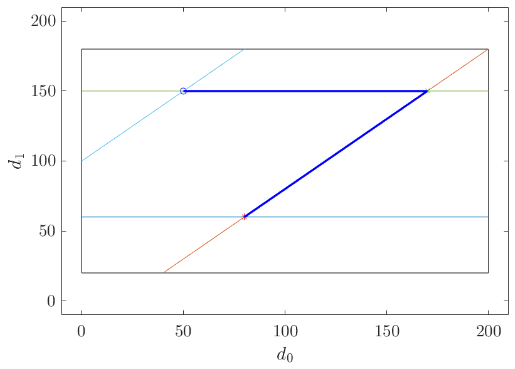

Fig. 6. A path search in 2D. The starting configuration is $(50, 150)$ and the target configuration is $(80, 60)$. The search suggests a two-step solution (blue), that passes through $(170, 150)$.

start configuration as the first node. The generalized search method is described in Algorithm 1.

C. Integration of the Adjustment Protocol

After understanding the inherent simplicity of the angle adjustment procedure, and the analytic nature of the fingertip distance adjustment procedure, we endeavor to combine the two. The combination of the two procedures is centered on time efficiency. Therefore, we seek a solution that will minimize any wastes of time in reconfiguration. To this end, we offer the following integration policy, that is *not* the optimal solution.

We prioritize fingertip distance adjustment over angle adjustment, given that some angular adjustments can be performed simultaneously to finger distance adjustment, and are likely to be possible. This is a purely statistical approach. Given n dimensions, if we prioritize angular adjustment, there is only a $1/n$ change that the finger being adjusted angularly corresponds with the next motion vector in the fingertip distance change path. Conversely, if we prioritize fingertip distance change, the probability that the pressed fingertip can also be angularly reconfigurated is *at worst* $1/n$. Therefore, prioritizing distance reconfiguration, and changing finger angles as the opportunity arises is more time-efficient than vice-versa.

The integration of the adjustment protocol is therefore quite simple. We follow the fingertip distance adjustment procedure as-is. When a fingertip is pressed, we check if it can be rotated to its angular target. If so, distance and angular reconfiguration are performed simultaneously. If not, only fingertip distance is changed. After completing all distance changes, we check if all angle changes have been completed. If not, we perform the remaining angle adjustments by sequential first-possible first-change methodology (brute force).

In the future, we will attempt to blend the angular and fingertip distance procedures into a singular time-domain. In this way, we will search for a time-optimal path that resolves both fingertip distance an finger angle problems simultaneously.

Notes for Algorithm 1:

Algorithm 1 Distance Adjustment Procedure

```
1: Initialization:
2: $n \leftarrow$ number of digits
3: $d^S = [d_0^S, \dots, d_{n-1}^S] \leftarrow$ initial digit distances
4: $d^T = [d_0^T, \dots, d_{n-1}^T] \leftarrow$ target digit distances
5: $L = [l_0^{min}, l_0^{max}, \dots, l_{n-1}^{min}, l_{n-1}^{max}] \leftarrow$ physical limits
6: $\mathbf{m} = m_1 \dots m_n \leftarrow$ CREATE_MOVEMENT_VECTORS($n$)
7: Cuboid $\leftarrow$ CREATE_CUBOID($L, \mathbf{m}$)
8: HP $\leftarrow$ EMANATE_HYPERPLANES($d^T, \mathbf{m}$)
9: Execution:
10: Path $\leftarrow$ BFS($d^S, d^T, \mathbf{m}$, Cuboid, HP)
11:
12: function CREATE_MOVEMENT_VECTORS($n$)
13:     $M_{n \times n} \leftarrow$ ones
14:     $M_{2..n \times 2..n} \leftarrow M_{2..n \times 2..n} - I_{n-1 \times n-1}$
15:     $\mathbf{m} \leftarrow$ rows of $M$
16:     return $\mathbf{m}$
17: function CREATE_CUBOID($L, \mathbf{m}$)
18:     $Corner_{min} \leftarrow [l_0^{min}, l_1^{min}, \dots, l_{n-1}^{min}]$
19:     $Corner_{max} \leftarrow [l_0^{max}, l_1^{max}, \dots, l_{n-1}^{max}]$
20:     $\mathbf{facets}_{min} \leftarrow$ null($Corner_{min}, \mathbf{m}$)
21:     $\mathbf{facets}_{max} \leftarrow$ null($Corner_{max}, \mathbf{m}$)
22:     Cuboid_Facets $\leftarrow \mathbf{facets}_{min} \cup \mathbf{facets}_{max}$
23:     return Cuboid_Facets
24: function EMANATE_HYPERPLANES($d^T, \mathbf{m}$)
25:     $\mathbf{hp}_0 \leftarrow d^T$
26:     for $i$ in $1 \dots n-1$ do
27:         $\mathbf{hp}_i \leftarrow$ null($\mathbf{hp}_{i-1}, \mathbf{m}$)
28:     HP $\leftarrow \mathbf{hp}_0 \dots \mathbf{hp}_{n-1}$
29:     return HP
30: function BFS($d^S, d^T, \mathbf{m}$, Cuboid, HP)
31:     Q $\leftarrow d^S$
32:     $\mathbf{m}'_0 \leftarrow \mathbf{m}$
33:     $i \leftarrow 0$
34:     for $v$ in Q do
35:         HPP,FP $\leftarrow$ EXPAND($v, \mathbf{m}'_i$, Cuboid, HP)
36:         for $w$ in HPP do
37:             $PathEnd \leftarrow$ LAST_STEPS_SEARCH($w$)
38:             if $PathEnd \neq$null then
39:                 return RETRACE_PATH($w$)$\cup PathEnd$
40:                 break
41:         Q $\leftarrow$ Q $\cup$ FP
42:     $i \leftarrow i + 1$
43: function EXPAND($v, \mathbf{m}'$, Cuboid, HP)
44:     lines $\leftarrow v + \mathbf{m}'$
45:     Facet_Pierces $\leftarrow$ Coincidence(lines, Cuboid)
46:     Pot_HP_Pierces $\leftarrow$ Coincidence(lines, HP.hp$_{n-1}$)
47:     HP_Pierces $\leftarrow$ Pot_HP_Pierces in Cuboid
48:     return HP_Pierces, Facet_Pierces
49: function LAST_STEPS_SEARCH($v$, Cuboid, HP)
50:     This function will provide a path from a point on a hyperplane to the target, if one exists
51: function RETRACE_PATH($v$)
52:     if $v = d^S$ then
53:         return $v$
54:     else
55:         Path $\leftarrow$ Path $\cup$ RETRACE_PATH($v.parent$)
56:     return Path
```

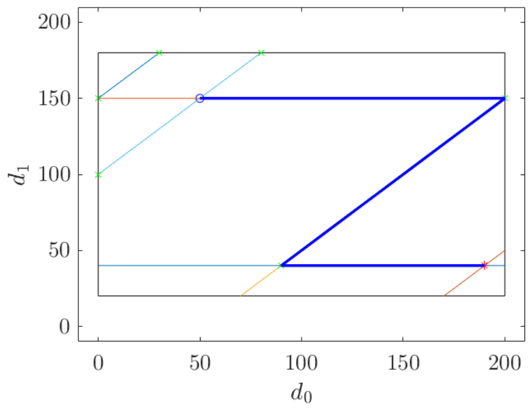

Fig. 7. Another path search in 2D. This time, the algorithm could not find a two-step solution, but could find a three-step one.

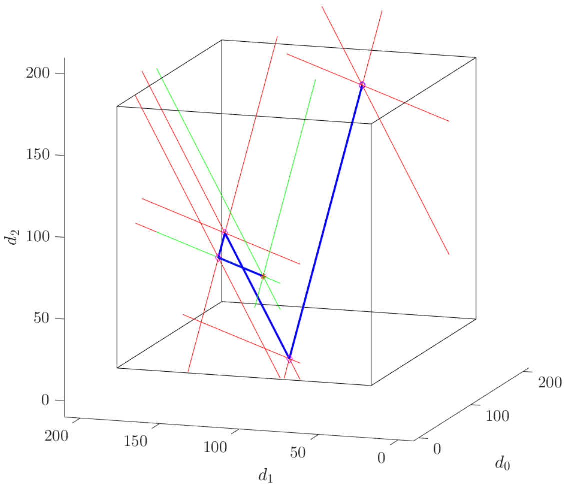

Fig. 8. A path search in 3D. Starting from (139,71.5,172) and ending at (6,90,81), a path with four steps was found. The red and green lines indicate the movement vectors emanated from each vertex along the path.

- Variables in **boldface** are lists of similar items. For instance, **m** in line 6 is a list of vectors $m_1, m_2, \dots, m_n$. Vectors are not indicated by boldface.
- Matrices are indicated by subscripts with their dimensions, such as in line 14.
- The matrix I in line 14 is the identity matrix.
- The operation null(a, b) in lines 20, 21 and 27 is the nullspace of the matrix composed of a and b.
- The operation Coincidence(a, b) returns the vertex of coincidence between the one-dimensional line a in n-dimensional space with the $n-1$ dimensional hyperplane b in the same space. If the line is parallel to the hyperplane, the operation returns nothing.
- The notation $\mathbf{m}'$ in the EXPAND function (line 43) indicates the movement vector list, excluding the movement vector leading to the specific vertex v. This is to prevent retracing steps, and entering infinite loops.

V. EXPERIMENTS

In this section, we detail the real-world experiments performed with a three-finger hand. The experimental setup, both real and simulated, is depicted in Fig. 9. The hardware used in our experiments is as followed:

- A configurable robot hand with 3 digits.
- A Motoman UP6 6-DOF robotic arm.
- A Logitech C310 webcam
- A PC with an Intel i7-3770 3.4 GHz processor, 32 GB of RAM, 64 Bit system running Matlab 2019
- Assorted objects from the YCB object set [13]

In order to demonstrate the capabilities of the hand, we performed a number of full and partial grasp procedures on real-world objects. A non-physical simulation environment was constructed in the RoboDK robotic simulator. The simulation environment mimics the real-world environment to an extent, containing the robotic arm, hand, and surrounding objects. Objects from the YCB set can also be imported into the simulation, as their 3D models are provided by the object set creators [13].

A full grasp procedure is as such:

1) The robot hand is set at an arbitrary configuration, usually the last configuration used.
2) An item is placed within the pickup zone on the ground near the robot.
3) The robot photographs the object, and attains its shape as a polygon.
4) The grasp location and adjustment procedure are found, using the methods detailed in Section III.
5) The robot executes the hand readjustment procedure, using either a nearby wall, or the robotic arm base (user's choice).
6) The hand is manipulated towards the object, and grasps it without force feedback.
7) The object is manipulated towards a drop-off point, where it is placed.

A full grasp procedure can be seen in the video accompanying this paper. A partial grasp procedure is the same as a full grasp procedure, but without the physical act of grasping and manipulation (the last two items). Full grasp procedures were performed on the following objects in the YCB object set: Mustard bottle (object #6), Tuna fish can (object #7), Power drill (object #35). Partial grasp procedures were performed on objects #4,11,12,13,21,22,29,33,51,52,76,77. All grasp procedures were performed using three fingers. Three different types of fingertips were used, that differ only in their minimal and maximal distances from the hand center L_{min}, L_{max}. Fingertips were selected based on object size.

A. Representing the Object as a Polygon

In order to apply the methods of grasp selection described in Section III, we first must convert our previously unknown object to a polygon. To do this, the robotic arm has a simple RGB camera (webcam) mounted on it. The webcam is manipulated above the object. An image of the object is taken by the camera. The image is contrast balanced and converted to black and white. Noise is removed via a "close" morphology operator, along with several other standard image processing operations to clean the image. The object is now represented as a single "blob". The object is converted to a configuration space object by using an "erode" morphology operator. The kernel of the erode operator is a disk with the radius of the fingertips. The blob's perimeter is then taken as a list of pixel locations. Since this list is finite, the blob is inherently represented as a polygon. This polygon is reduced in its number of edges, according to a tolerance set by the user. A lower tolerance results in a better approximation to the original blob, and a higher number of edges. A low tolerance may result in undesirable "sharp edges", which misrepresent the object's surface normal. A higher tolerance results in a worse approximation of the blob, and a lower number of edges. The remainder of our algorithm is indifferent to the number of polygon edges, therefore the tolerance is set by trial and error, avoiding oversimplification one one hand, and artificial sharp edges on the other. We apply a transformation between camera coordinates and real-world coordinates, based on prior camera calibration. The final polygon is then passed on to the next portion of the procedure– grasp selection.

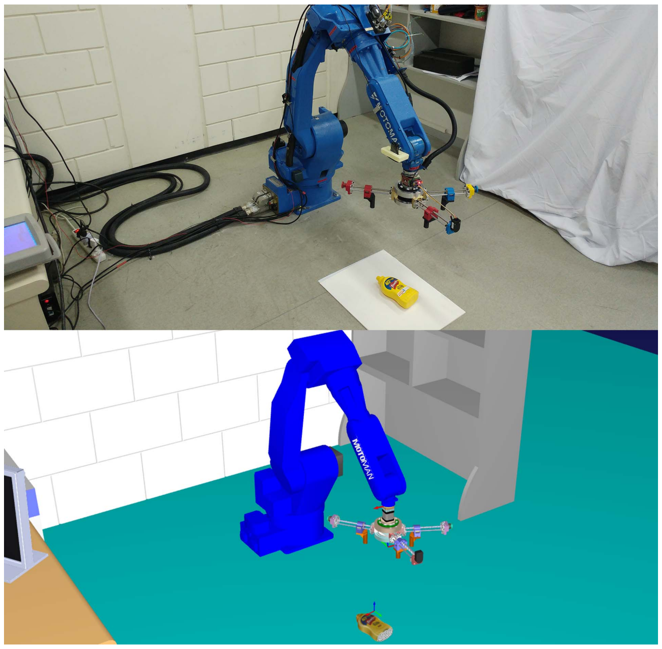

Fig. 9. The experimental setup. The real-world experiments (top) mirror the simulation environment (bottom). A "Mustard Bottle" item is in the pickup zone.

B. Execution of the Adjustment Procedure

Following grasp selection (Section III) and adjustment procedure synthesis (Section IV), a list of instructions is provided. This list is, in fact, the adjustment procedure. When followed, this list of instructions transforms the hand from its initial configuration to the target configuration suited to grasp the object. An Arduino Nano controls the stepper motor in the robot hand. Rotation validation is obtained by a rotary encoder fixed to the the thumb. The instructions can be carried out in simulation mode, or in *run-on-robot* mode. For safety reasons, experiments were carried out in simulation mode, before being replicated in run-on-robot mode.

VI. Conclusion

In this work, we introduced a novel robot hand, and the motion planning algorithm used to reconfigure it. The main contribution of this report is the introduction of the finger distance adjustment procedure. The secondary contribution is the proof of simplicity in the angle adjustment procedure. The motion planning principles presented are not limited to the robot hand presented, and can be expanded to other hands or completely different problems. For example, the fingertip distance function can be used for robot navigation in a bounded space with movement constraints.

In our real world experiments, three objects were grasped and manipulated using the full grasp procedure, and 12 additional objects were used for partial grasp procedures. In all 15 cases, the system was able to find grasp configurations, and synthesize the adjustment procedure. The adjustment procedure was executed for each of the cases successfully. Objects #11, 12 and 77 are relatively small, therefore fingertips with a small minimal distance were used. Similarly, fingertips with a large maximal distance were used for the larger object #33. The remainder of objects were grasped using mid-range fingertips.

In the three manipulation tasks, both the tuna can and mustard bottle were manipulated successfully. The power drill was grasped, but slipped from the hand during manipulation. This is because the grasp force was relatively low by design, as part of the experimental safety measures. The force exerted by each finger is at most 5 N, which was insufficient to vertically manipulate the heavier power drill without slipping. The next step in our experiments is to relax the safety restrictions, increasing grasp force and re-adjustment time.

This technical report serves as supplementary theoretical background for related works. In the future, the theoretical principles presented here will be expanded and published. At that point this report shall be deprecated.

References

[1] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, and K. Goldberg, "Learning ambidextrous robot grasping policies," *Science Robotics*, vol. 4, no. 26, p. eaau4984, 2019.

[2] S. B. Backus and A. M. Dollar, "An adaptive three-fingered prismatic gripper with passive rotational joints," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 668–675, 2016.

[3] A. M. Dollar and R. D. Howe, "A robust compliant grasper via shape deposition manufacturing," *IEEE/ASME transactions on mechatronics*, vol. 11, no. 2, pp. 154–161, 2006.

[4] J. Shintake, V. Cacucciolo, D. Floreano, and H. Shea, "Soft robotic grippers," *Advanced Materials*, vol. 30, no. 29, p. 1707035, 2018.

[5] S. Terryn, J. Brancart, D. Lefeber, G. Van Assche, and B. Vanderborght, "Self-healing soft pneumatic robots," *Sci. Robot*, vol. 2, no. 9, pp. 1–12, 2017.

[6] F. Ilievski, A. D. Mazzeo, R. F. Shepherd, X. Chen, and G. M. Whitesides, "Soft robotics for chemists," *Angewandte Chemie International Edition*, vol. 50, no. 8, pp. 1890–1895, 2011.

[7] E. Brown, N. Rodenberg, J. Amend, A. Mozeika, E. Steltz, M. R. Zakin, H. Lipson, and H. M. Jaeger, "Universal robotic gripper based on the jamming of granular material," *Proceedings of the National Academy of Sciences*, vol. 107, no. 44, pp. 18 809–18 814, 2010.

[8] R. Deimel and O. Brock, "A novel type of compliant and underactuated robotic hand for dexterous grasping," *The International Journal of Robotics Research*, vol. 35, no. 1-3, pp. 161–185, 2016.

[9] Y. Golan, A. Shapiro, and E. Rimon, "Jamming-free immobilizing grasps using dual-friction robotic fingertips," *IEEE Robotics and Automation Letters*, 2020, preprint.

[10] SMC Inc. Slide Guide Round Body Air Gripper 3-Finger Type MHS3. (2020, Feb. 15). [Online]. Available: https://www.smcin.com/content/slide-guide-round-body-air-gripper-3-finger-type-mhs3

[11] E. Rimon and J. Burdick, *The Mechanics of Robot Grasping*. Cambridge University Press, 2019.

[12] H. A. Bunis, E. D. Rimon, Y. Golan, and A. Shapiro, "Caging polygonal objects using formationally similar three-finger hands," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3271–3278, 2018.

[13] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.