Coordinating a Robot Arm and Multi-finger Hand Using the Quaternion Representation

Kenneth S. Roberts

Department of Computer Science
Columbia University
New York, New York 10027
roberts@cs.columbia.edu

Technical Report CUCS-481-89

Coordinating a Robot Arm and Multi-finger Hand Using the Quaternion Representation

Kenneth S. Roberts[1]

Department of Computer Science
450 Computer Science Building
Columbia University
New York, NY 10027
roberts@cs.columbia.edu

Technical Report CUCS-481-89
October 1989

Abstract

An autonomous hand-arm system must be able to move its multiple fingertips to specified 3-D world locations. We present an algorithm which, given a desired position and normal for each fingertip, computes all the joint angles for the fingers and arm. Our primary method for handling this underconstrained problem is to optimize a cost function. We also give methods for generating good candidates to be optimized. We present several new techniques for using the quaternion form to optimize rotation. We give experimental results from using the algorithm to apply complicated grasps with a Utah/MIT hand-arm system.

1 Introduction

An autonomous hand-arm system must be able to move its multiple fingertips to specified world locations. The difficulty is that a robot hand-arm system has many degrees of freedom. (Our Utah/MIT hand-arm system has 4 fingers with 4 joints each, plus a 6-joint arm, for a total of 22.) We present an algorithm to coordinate all those joints to reach several locations, or to say if those cannot all be reached simultaneously.

Here are two situations in which this algorithm would be useful. First, for grasping: Assume that an accurate geometric model for the object is known, and that

[1]The author is an AT&T Bell Laboratories PhD Scholar. This work was also supported in part by DARPA contract N00039-84-C-0165, NSF grants DMC-86-05065, DCI-86-08845, CCR-86-12709, IRI-86-57151, North American Philips Laboratories, and the AT&T Foundation.

the current location of the object is accurately known from sensing. A grasp planner, working from the task requirements and the object geometry, has proposed a configuration of contact positions and normals to stably grasp the object. Possibly this configuration was pre-computed off-line. The problem is how to actually reach this configuration, on the object in this location, with this multi-finger hand-arm manipulator.

Second, for exploration: Suppose we are doing model-based recognition and localization, and are working from a current hypothesis about the identity and location of the object. A tactile sensing planner has identified several positions on the surface of the hypothesized object which should be explored next, along with surface normals which reflect preferred sensing orientations for tactile sensors on the fingertips. The problem now is how to reach these locations simultaneously with this multi-finger hand, assuming that the object is in the hypothesized location.

1.1 Problem statement

The multi-finger robot hand has n fingers, and finger f has m_f joints. θ_{fj} is the angle of joint j on finger f. The hand is supported by a 6 degree of freedom robot arm, with 6 angles ϕ_i. We are given a desired position p_f^d and outward normal n_f^d for the fingertips of some of the n fingers. We assume that contact is made not with the most distal point of the tip, but with the front "pad" surface of the fingertip link, so that the surface normal is perpendicular to the axis of the link (see Figure 1). We are also given weights of the relative importance of each of these positions and each of these normals. We are to find the $\sum m_f$ finger joint angles θ_{fj} and 6 arm angles ϕ_i which reach those locations (or near to them).

For the remainder of this paper, we will explain everything in terms of the specifics of our multi-finger hand-arm system at Columbia. For our Utah/MIT hand, $n = 4$ and $m_f = 4$. Its arm is the Utah/MIT remotizer unit. See Appendix A for more on the kinematics of the fingers, hand, and arm.

For each single finger, there is a chain of 4 finger joints, and then 6 arm joints, from the fingertip back to the base of the arm. If we are given the desired position and normal at that single fingertip, that yields only 5 independent constraints – so the 10 degrees of freedom are redundant. Even if the positions and normals of all 4 fingers are given, the problem is still underconstrained: 20 constraints vs 22 degrees of freedom.

1.2 Previous work

Tomovic *et al.* [1987] (see also [Rao *et al.*, 1988]) and Stansfield [1989][1988] have presented multi-finger hand-arm systems for grasping an object whose shape model

is acquired from vision, rather than a previously-known CAD database. The grasp is chosen by a rule-based reasoning from a small family of generic grasp types. Some simple grasp parameters are calculated from the acquired object shape (e.g., how wide to open the fingers, how orient the axis of the grasp), but it is not the intention to try to place specific hand contact areas onto particular points on the object. This approach seems sensible and promising, and our algorithm could be used to take the generic grasp result from one of their systems, and optimize it for the detailed geometry of this specific object and the reachability difficulties of this specific location.

The Handey system [Lozano-Perez *et al.*, 1987] uses detailed geometric knowledge to synthesize a grasp, and also deals with path-planning and collision avoidance (which we have not). But Handey uses only a parallel-jaw gripper, not a multi-finger hand.

Li and Sastry [1988] (also [Li *et al.*, 1989]) and Grupen and Henderson [1988] have presented methods for finding optimal multi-finger grasps. Their emphasis is on incorporating task information, so we see our focus on kinematic reachability as complementing theirs. Possibly their approach could be used to pre-compute off-line an optimal grasp configuration of positions and normals for a particular object and task, and then our procedure could find the optimal way to reach it after the specific location has been acquired by sensing.

The organ-playing WABOT-2 [Sugano and Kato, 1987] produced smooth finger and arm trajectories from a musical score, and also chose which finger should play each note. But it could successfully play a keyboard musical instrument without changing the rotation of the wrist, and thus avoided the major concern of our paper.

Nguyen [1988][1989] has worked on synthesizing grasps for a known object, but with the goal of grasp stability, not reachability. Our algorithm complements this very nicely.

Bay [1989] discusses hand-arm coordination in connection with his work on multi-finger exploration, but only deals with incremental local movements along the surface of the object, and does not consider the problem of reaching a set of specified positions.

2 Finding the optimal arm position

2.1 The cost function

Our primary method for dealing with this underconstrained problem is to minimize a cost function. Our cost function is

$$\begin{aligned}
&\sum_{i=0}^{5} w_{\phi i}(\phi_i - \check{\phi}_i)^2 \\
&+ \sum_{f=0}^{3}\sum_{j=0}^{3} w_{\theta fj}(\theta_{fj} - \check{\theta}_{fj})^2 \\
&+ \sum_{f=0}^{3} w_{Pf}(\mathrm{p}_f - \mathrm{p}_f^d)\cdot(\mathrm{p}_f - \mathrm{p}_f^d) \\
&+ \sum_{f=0}^{3} 2w_{Nf}(1 - \mathrm{n}_f \cdot \mathrm{n}_f^d)
\end{aligned}$$

The first term says that we want to stay away from the limit stops on the arm joint angles. Even if an arm configuration near the limits is legal for the initial grasp, it may be violated by small task-directed manipulations that follow, so there is some value to staying away from the limits. We set $\check{\phi}_i$ to the midpoint between the upper and lower limits. We set weights $w_{\phi i}$ proportional to the reciprocal of the square of the difference between the limits. If an arm configuration actually violates the limits, we set the cost function to a very large value.

The second term says that we want to stay away from the limit stops on the finger joints. We set $\check{\theta}_i$ to the midpoint between the upper and lower limits. We set weights $w_{\theta fj}$ proportional to the reciprocal of the square of the difference between the limits. In our procedure, the proposed θ_{fj} values are always chosen so that they cannot violate the limit stops.

The third term says that we want to achieve the desired positions, by minimizing the distance between the actual position p_f and the desired position p_f^d. By the forward kinematic formulas, p_f is a function of the actual joint angles $(\theta_{f0}, \theta_{f1}, \theta_{f2}, \theta_{f3})$ (see Appendix A). The weights w_{Pf} are supplied by the task, and permit us to focus the accuracy on a subset of the fingers.

The fourth term says that we want to achieve the desired normals. It is an approximation to γ^2, where γ is the angle between the actual and the desired normals, and is derived from $\mathrm{n}_f \cdot \mathrm{n}_f^d = \cos\gamma \simeq 1 - \gamma^2/2$. (The use of this approximation for γ^2 is new, as far as we know, at least in the robotics community. Bay [1989] uses a very different measure). By the forward kinematic formulas, the actual normal n_f is a function of the actual joint angles $(\theta_{f0}, \theta_{f1}, \theta_{f2}, \theta_{f3})$ (see Appendix A).

(If a desired fingertip orientation were specified as a complete rotation matrix for the fingertip coordinate frame, a simple error cost measure would be $8w(1-\bar{\mathrm{q}}\cdot\bar{\mathrm{q}}^d)$, where $\bar{\mathrm{q}}^d$ is the quaternion form of the desired rotation, discussed further in section 5.2.1).

2.2 Choice of independent parameters

The cost function is expressed in terms of the 16 finger joint angles θ_{fj} and the 6 arm angles ϕ_i But these cannot all be independent parameters, which raises the question of what set of parameters to use for optimization.

Physically, there are 6 degrees of freedom for the whole hand, 3 for the rotation and 3 for the translation of the hand coordinate frame transform. So we want to have 6 independent parameters to optimize. An obvious choice would be the 6 arm angles ϕ_i. One disadvantage for these is that their derivatives interact in complicated non-symmetrical ways. Another is that we might decide later not to include the arm angles in the cost function at all. and instead just use the arm kinematics to test for reachability — in which case it is a waste of computation to use them as our fundamental parameters.

Instead, we have chosen to optimize on the rotation and translation parameters themselves: three quaternion parameters $\mathrm{q} = (q_x, q_y, q_z)$ for the rotation (explained below in section 3), and the position vector $\mathrm{t} = (t_x, t_y, t_z)$ for the translation of the hand frame. We can calculate the hand frame's 4×4 homogeneous transform $\mathbf{H}_{w \leftarrow h}$ from these by converting the three quaternion parameters into the 3×3 rotation matrix component (see section 3.2) and incorporating t in the usual way. From $\mathbf{H}_{w \leftarrow h}$ we use the finger and arm inverse kinematic formulas to get θ_{fj} and ϕ_i, the inputs to our cost function above. If the hand transform is not reachable by the arm kinematics, we set the cost function to an arbitrary large value. But if a fingertip location is not reachable by the finger kinematics, then we substitute a nearby fingertip location which is reachable, and the cost function will reflect the amount of the discrepancy.

3 Quaternion form for rotation

3.1 Choice of representation

The first major choice in dealing with rotation is what representation to use.

The common method in robotics is to use a 3×3 matrix. This form is usually the most convenient for transforming vectors, computing kinematic solutions, etc. For intermediate computations, it has the obvious disadvantage of needing more memory space; and it uses more operations for calculations such as composing two rotations. For optimization, the fundamental difficulty with the matrix form is that

the 9 parameters not independent. They are subject to 6 orthogonality constraints. Also, if a matrix does not exactly satisfy those orthogonality constraints (due to numerical errors, etc), how to re-normalize it correctly is not obvious or simple. And even if a given matrix satisfies the constraints, it may be an improper rotation (determinant = -1), and thus impossible for a rigid body.

A 3-D rotation has 3 degrees of freedom, so it is natural to seek a 3-parameter representation, such as the well-known Euler angles (of which yaw-pitch-roll is an instance). But Euler angles have some important disadvantages: Computing with them can be very cumbersome and often requires evaluating many trig functions. Also, there are singularities in the representation at certain configurations of angles (which can be observed physically as "gimbal lock"). We consider other 3-parameter representations in section 3.3.4.

The quaternion[2] form for a rotation has 4 parameters. It is true that this is one more than the required 3, but recall that when representing a 3-D position it is often convenient to add a 4th homogeneous coordinate. If we have a rotation by an angle θ about an axis given by the unit vector $\mathbf{s} = (s_x, s_y, s_z)$, then the quaternion form is:

$$\begin{aligned} \bar{\mathbf{q}} &= [q_0, \mathbf{q}] \\ &= [q_0, q_x, q_y, q_z] \\ &= \left[\cos\frac{\theta}{2}, s_x \sin\frac{\theta}{2}, s_y \sin\frac{\theta}{2}, s_z \sin\frac{\theta}{2}\right] \end{aligned}$$

By "quaternion", we mean a unit quaternion, which has unit magnitude ($q_0^2+q_x^2+q_y^2+q_z^2 = 1$). This is the only kind anyone uses when dealing with rotations. q_0 is called the scalar component, and $\mathbf{q} = (q_x, q_y, q_z)$ the vector component. ($\mathbf{q} = \sin(\theta/2)\mathbf{s}$). So the set of (unit) quaternions forms a unit 3-sphere in R^4. Each rotation is then represented by a pair of points on the quaternion sphere, because $+\bar{\mathbf{q}}$ and $-\bar{\mathbf{q}}$ are the same rotation.

The quaternion form does have the disadvantage of an extra parameter, but it has many beautiful properties, which are well presented elsewhere [Goldstein, 1980] [Spring, 1986] [Horn, 1987] [Shoemake, 1987]. Unlike Euler angles, it has no singularities (beyond the omnipresent sign ambiguity just described). The quaternion form provides a nice uniform metric: the arc distance between two points on the quaternion sphere is exactly half the rotational angle needed to transform the one rotation into the other (see section 5.2.1). Also, it is fast to compute with, especially when composing two rotations, or converting between quaternion and matrix form.

[2]Spring [1986] (also [Goldstein, 1980] and others) points out that this 4-parameter form was invented by Euler in 1776, while Hamilton invented the quaternion algebra in 1843 so that the historically more correct term is "Euler parameters". Nevertheless, the quaternion division operation is valuable for dealing with rotations, and the term "quaternion" avoids confusion with "Euler angles".

Unlike a matrix, a quaternion is easy to re-normalize when it fails to satisfy the unit magnitude constraint (due to numerical errors, etc) (see [Funda and Paul, 1988]).

Physicists have long known that the quaternion form is good for handling rotational motion [Goldstein, 1980], and it has been found useful for controlling spacecraft. Several researchers have used it for robotics [Chou and Kamel, 1988] [Lin, 1988] [Bay, 1989] and for computer vision and graphics problems [Horn, 1987] [Roberts *et al.*, 1988] [Shoemake, 1987].

3.2 Some quaternion calculations

The formula for composing two rotations in quaternion form is

$$\bar{\mathrm{q}}\,\bar{\mathrm{r}} = [q_0, \mathrm{q}][r_0, \mathrm{r}] = [q_0 r_0 - \mathrm{q}\cdot\mathrm{r}, \quad r_0\mathrm{q} + q_0\mathrm{r} + \mathrm{q}\times\mathrm{r}]$$

The identity rotation has a rotational angle $\theta = 0$:

$$[\text{identity}] \quad = \quad [1,0,0.0] \quad \text{or} \quad [-1,0,0,0]$$

The quaternion inverse is

$$\bar{\mathrm{q}}^{-1} = [q_0, \mathrm{q}]^{-1} = [q_0, -\mathrm{q}]$$

A quaternion $\bar{\mathrm{q}}$ may rotate a 3-D position vector a by pre- and post-multiplying by the quaternion and its inverse:

$$\bar{\mathrm{q}}\,\mathrm{a}\,\bar{\mathrm{q}}^{-1} = \bar{\mathrm{q}}\,[0, \mathrm{a}]\,\bar{\mathrm{q}}^{-1} = [q_0, \mathrm{q}][0, \mathrm{a}][q_0, -\mathrm{q}]$$

This has the effect of actively rotating a about the axis q in the right-handed sense, by the angle $\theta = 2\arccos q_0$. Carrying out the calculation shows that the 3×3 matrix form of the quaternion $[q_0, q_x, q_y, q_z]$ is

$$\begin{bmatrix} (q_0^2 + q_x^2 - q_y^2 - q_z^2) & 2(q_x q_y - q_0 q_z) & 2(q_x q_z + q_0 q_y) \\ 2(q_y q_x + q_0 q_z) & (q_0^2 - q_x^2 + q_y^2 - q_z^2) & 2(q_y q_z - q_0 q_x) \\ 2(q_z q_x - q_0 q_y) & 2(q_z q_y + q_0 q_z) & (q_0^2 - q_x^2 - q_y^2 + q_z^2) \end{bmatrix}$$

The inverse problem of finding the quaternion form of a given 3×3 rotation matrix is straightforward, but care must be taken to ensure numerical stability. The best procedure we know of is in [Horn, 1987].

3.3 Using quaternions for optimization

Our problem is to find a quaternion that optimizes some cost function. We are concerned here with the usual case of non-linear optimization by iterative improvement [Press *et al.*, 1988] [Gill *et al.*, 1981]. (although sometimes a rotation problem can

be solved by linear least-squares, as in the beautiful result of Horn [1987]). At each step of the iteration, a new value for the parameter vector is generated, and the cost function (and perhaps also some derivatives) is evaluated on that parameter vector.

The difficulty is that we have more parameters than the degrees of freedom: 3 degrees of freedom versus 4 quaternion parameters with 1 constraint equation. There are several ways to handle this:

1. Ignore the constraint during optimization. then normalize the final result.
2. At each iteration, initially generate an unconstrained parameter vector, but normalize it before evaluating the cost function and generating another parameter vector.
3. Add a "constraint violation" term to the cost function: $k_q(1 - \bar{\mathbf{q}} \cdot \bar{\mathbf{q}})^2$. As the iterations continue, the value of k_q is increased, so that by the end of the iterations, the constraint is nearly satisfied.
4. Use only 3 parameters: $\mathbf{q} = (q_x, q_y, q_z)$. Then $q_0 = (1 - \mathbf{q} \cdot \mathbf{q})^{1/2}$. This requires an inequality constraint $\|\mathbf{q}\| \leq 1$.

3.3.1 Totally unconstrained

The difficulty here is that the quaternion parameters must be normalized before they can be applied to rotate a 3-D position vector, or generate a valid rotation matrix, which we must do in order to evaluate the cost function. So there is no advantage to ignoring the constraint when generating new quaternion parameters.

3.3.2 Impose constraint at each iteration

The idea here is to generate the new parameter vector using standard unconstrained iteration techniques, and then normalize it before evaluating the cost function and generating the next seed. This is simple. but has some disadvantages:

- The computational time and memory cost of carrying the redundant parameter at all times.
- For optimization methods not using derivatives, it is desirable to maintain a set of linearly independent directions in which to move in the parameter space. But here, directions which appear independent (e.g. $[-1, 0, 0, 0]$ and $[0, 0.577, 0.577, 0.577]$) are not, since the quaternions that result after those steps are applied will be identical, once they have been normalized.

- For optimization methods using derivatives, the difficulty is that quantities based on the parameters (which will include some components of the cost function) may be expressed in more than one possible way. E.g. the component R_{00} of the rotation matrix R can be expressed as $(q_0^2 + q_x^2 - q_y^2 - q_z^2) / (q_0^2 + q_x^2 + q_y^2 + q_z^2)^{(1/2)}$ or as $(1 - 2q_y^2 - 2q_z^2) / (q_0^2 + q_x^2 + q_y^2 + q_z^2)^{(1/2)}$ Depending on which one is chosen, there is a different value for the derivative dR_{00}/dq_x. There is also the difficulty of avoiding singularities due to division by zero in calculating derivatives, and linear dependence in calculating matrix inverses.

3.3.3 Add a "constraint violation" cost

The idea here is to actually use the parameters without normalizing, but to require that they get closer and closer to exactly normalized as the number of iterations increases. The advantage is again simplicity: just modify the cost function and then use standard unconstrained optimization methods.

The problem is that the matrix R that results from an un-normalized quaternion is not a pure rotation. It includes a scaling factor of $\|\bar{q}\|$. For most applications, this scaling effect is physically forbidden. It will likely introduce a new set of local minima which will confuse the global search.

3.3.4 Use only 3 parameters

The 3 parameters are (q_x, q_y, q_z), the quaternion's vector component $\mathbf{q} = \sin(\theta/2)\mathbf{s}$, where s is the axis and θ the angle of the rotation. From these, we may compute the other parameter: $q_0 = (1 - \mathbf{q} \cdot \mathbf{q})^{1/2} = (1 - q_x^2 - q_y^2 - q_z^2)^{1/2}$. Our convention is to choose the positive square root. The 3 parameters are subject to an inequality constraint, $\|\mathbf{q}\| \leq 1$. As far as we know, this is the first time this has been used as a 3-parameter representation for rotations.

There are other possibilities. We already discussed the well-known Euler angles in section 3.1. The Rodrigues parameters (or Gibbs vector) are $\tan(\theta/2)\mathbf{s}$, but these blow up near $\theta = \pm\pi$. Gu [1988] uses a Lie algebra approach to derive some 3-parameter representations, and proposes θs, and also $\sin(\theta)$s (the vector component of $[\cos(\theta), \sin(\theta)\mathbf{s}]$, sometimes called the "linear parameters"). These are only slightly different from the quaternion vector component form we are presenting here, but give up its advantages without good reason.

The formula for q_0 above gives an 1-to-1 mapping between the set of all proper rotations and the unit 3-ball $\{\|\mathbf{q}\| < 1\}$ (also including part, but not all, of the bounding unit 3-sphere). This is because, for every proper rotation whose angle is not $\pm\pi$, there is exactly one unit quaternion with $q_0 > 0$; so the 3 parameters are uniquely given by the vector component q of that quaternion. For a rotation whose

angle is $\pm\pi$, there are two corresponding unit quaternions, $[0, \mathbf{q}]$ and $[0, -\mathbf{q}]$. We can get uniqueness by imposing the following complicated condition: Choose the sign of q so that $q_x > 0$. If $q_x = 0$, choose so that $q_y > 0$. If $q_y = 0$, choose so that $q_z > 0$. This complicated condition defines a kind of hemisphere.

Let $\mathcal{B}$ be the set of unique 3-vectors: all vectors in the unit open 3-ball ($\|\mathbf{q}\| < 1$), along with all vectors on the unit 3-sphere ($\|\mathbf{q}\| = 1$) which satisfy the complicated condition at the end of the previous paragraph. So $\mathcal{B}$ is a unit ball with half its bounding sphere missing. And we have exhibited a 1-1 mapping from the set of proper rotations to the set $\mathcal{B}$.

Topology

While it is valuable to exhibit a 1-1 mapping of rotations into a space with 3 independent parameters, it is also important to examine the topology of this space. When searching a parameter space for an optimum, it is good to have local neighborhoods in the physical problem space (set of proper rotations) map to local neighborhoods in the parameter space (e.g. the unit quaternion sphere in R^4, or here the set $\mathcal{B}$ in R^3). This is because non-linear optimization takes place by local hill-climbing steps in the parameter space. If a neighborhood in rotation space maps to some disconnected set in the parameter space, then a local step which is physically reasonable will not be available to the optimization procedure, which may then get stuck. The parameter space may have more local minima than the physical problem space.

The topology (i.e. structure of neighborhoods — rigorously, of open sets) of a space depends on the distance measure, or "metric" chosen. The usual Euclidean metric for sets in R^n is $d(\mathbf{q}, \mathbf{r}) = \|\mathbf{q} - \mathbf{r}\|$. A point is in the "neighborhood" of another given point if the metric distance between them is less than some specified small value. There is a "homeomorphism" between two sets if there is an invertible mapping between them that maps the open neighborhoods of one set into open neighborhoods in the other set (in either direction). It is a theorem that a continuous mapping which has a continuous inverse exhibits a homeomorphism.

The proper rotations may be represented by the set of all 3×3 matrices which are orthogonal and have a positive determinant. In group theory, this set of matrices, with matrix multiplication as its group operation, is called SO(3) ("special orthogonal"). There is exactly one matrix in SO(3) for every physical rotation. We may define a "Euclidean" metric on SO(3) as $d(\mathbf{A}, \mathbf{B}) = \sum\sum(\mathbf{A}_{ij} - \mathbf{B}_{ij})^2$, which roughly corresponds to whether two matrices are similar or different physically.

With $\mathcal{Q}$, the set of unit quaternions, there is a complication since $\bar{\mathbf{q}}$ and $-\bar{\mathbf{q}}$ both map to the same rotation. So we can examine instead the set of antipodal pairs of quaternions (which is the same as the real projective 3-sphere). To maintain physical significance, we need to modify the Euclidean metric to: $d(\bar{\mathbf{q}}, \bar{\mathbf{r}}) = \min\{\|\bar{\mathbf{q}} - \bar{\mathbf{r}}\|, \|\bar{\mathbf{q}} + \bar{\mathbf{r}}\|\}$. Under these metrics, a simple examination of the mapping formulas is sufficient to verify that there is a homeomorphism between the set of proper rotations SO(3) and the pairs of unit quaternions $\mathcal{Q}$. If we insist on single points

instead of antipodal pairs, we can use only one "half" of the 3-sphere in R^4 [with some complications at the hemisphere boundary] with the same modified metric, and still get the desired homeomorphism). With the unmodified Euclidean metric, there is no homeomorphism between SO(3) and any subset of the single points of R^4, but there is such a subset of R^5 [Hopf, 1940][Stuelpnagel, 1964].

Finding a metric gets still more complicated for the set $\mathcal{B}$. $(0.999, 0, 0)$ and $(-0.999, 0, 0)$ both refer to rotations of roughly π about the x-axis, but the Euclidean distance between them in $\mathcal{B}$ is nearly 2. And the metric as modified for $\mathcal{Q}$ above does not work for $\mathcal{B}$, because it gives the distance between $(0.7, 0, 0)$ and $(-0.7, 0, 0)$ as 0, but those are entirely different rotations (one a rotation of roughly $\pi/2$, the other $-\pi/2$, about the x-axis). The solution is to permit the vector addition operation on $\mathcal{B}$ to "wrap around". Define a vector "wraparound addition" operator $\oplus$ as follows:

$$\mathbf{a} \oplus \mathbf{b} = \begin{cases} (\mathbf{a}+\mathbf{b}) & \text{if } (\mathbf{a}+\mathbf{b}) \in \mathcal{B} \\ (\mathbf{a}+\mathbf{b}) - 2\dfrac{(\mathbf{a}+\mathbf{b})}{\|\mathbf{a}+\mathbf{b}\|} & \text{otherwise} \end{cases}$$

This has the effect of translating the result through the origin by a Euclidean distance of 2. Now we can define a "wraparound" metric by

$$d(\mathbf{a}, \mathbf{b}) = \min_{\{\mathbf{c} \in R^3 | \mathbf{a} \oplus \mathbf{c} = \mathbf{b}\}}(\|\mathbf{c}\|)$$

Under this metric, it is evident that our mapping from the unit quaternions into the set $\mathcal{B}$ is a homeomorphism. So by composing mappings, there is a homeomorphism between $\mathcal{B}$ and the proper rotations SO(3). This will be a helpful property for optimization, provided that we take each local iterative step using the "wraparound addition" $\oplus$ operation. Under the unmodified Euclidean metric, Stuelpnagel [1964] gives a straightforward proof that there can be no homeomorphism between SO(3) and any subset of R^3.

Handling the inequality constraint in optimization

Our set of permitted vectors $\mathcal{B}$ is limited by the inequality $\|\mathbf{q}\| \leq 1$. There are several ways to deal with this:

1. If a parameter vector is generated that violates the constraint, simply reject it, or have the cost function return a ridiculously high value. As discussed above, this may result in needlessly getting stuck in a local minimum.

2. In some cases, it may be possible to set up the coordinate frame so that the places where the constraint applies are physically forbidden for some other reason. E.g. in our robot hand/wrist coordination problem, the joint angle limits in the arm kinematics make a large set of rotations impossible; so after we choose the coordinate frame carefully, there is no disadvantage in rejecting iterative steps which violate the inequality constraint.

3. Change the coordinate frame before every iteration, so that the resulting transformation of the parameter vector moves it to the origin, as far as possible from the inequality constraint. This is a lot of extra computation, and is inferior to the wraparound approach.

4. Use the "wraparound addition" operation when making the local iterative step. This gets around the inequality barrier problem, with very little additional computation.

We have implemented approach 4. But even with wraparound, it is a good idea to stay away from the $\|q\| = 1$ region where wraparounds occur. This is because some of the derivatives get very large around there, which could result in slow convergence or numerical instability. So it is still worthwhile to apply approach 2 and choose the coordinate frame carefully. Another possible technique is to count the wraparounds, and if too many go back and forth during a small number of iterations, then change the coordinate frame. (If we represent a rotation of angle θ about axis $s = (s_x, s_y, s_z)$ by the 3 parameters θs_x, θs_y, θs_z [Gu, 1988] we avoid these large derivatives, but give up some of the nice metric and computational properties of quaternions.)

4 Non-linear optimization technique

The cost function cannot be minimized in closed form or by linear least squares, so we must resort to non-linear optimization techniques. We have chosen Powell's method [Press *et al.*, 1988] [Acton, 1970], because it tends to generate "conjugate" directions (i.e. descending in one direction does not undo most of the previous gains from other directions), but without using derivatives. "No derivatives" makes initial implementation quick, makes changing the cost function quick, and makes it easy to introduce terms into the cost function whose derivatives are difficult or impossible to compute. The disadvantage of Powell's method is that it is slow. After it becomes more clear how the hand-arm coordination algorithm should fit into a larger task planning and execution system, it may be worthwhile to substitute a more efficient optimization technique.

Powell's method finds the direction of fastest descent by examining a sequence of roughly orthogonal sub-steps. So we do not "wrap around" the q parameters until the sequence of sub-steps is completed, so that the direction of fastest descent may accumulate undisturbed until end of the full iteration step.

Our algorithm has two termination criteria, which are checked after each iteration. If the cost function value is less than the "instant winner" level, stop and report success. If the fractional decrease from the previous iteration is less than a specified percentage (we use 0.001), stop and compare the value against the "success" level. If less, then report success, otherwise failure. The minimum decrease percentage is

fixed, but the the "instant winner" and "success" levels depend on the number of fingers concerned. The more fingers, the higher the levels, with the increase somewhat faster than linear (i.e. we relax the standards when there are more fingers). Our practice is to set the "instant winner" level equal to one-tenth of the "success" level.

5 Generating seeds

One of the most important things for using an iterative optimization technique is to choose a good seed or starting value. A bad seed may result in slow convergence, or convergence to a false local minimum. Our approach is to generate several candidate seeds, evaluate the cost function on each one, then optimize the best candidate.

5.1 Seed for a single finger

We first take the desired position and normal for a single fingertip, and impose additional constraints so that we can calculate a unique value for our 6 hand transform parameters which reaches that desired location. We start with a preferred y-axis and a preferred x-axis for the hand frame (see below on where these come from). We rotate these into the corresponding axes in the local finger coordinate frame, so we have a preferred finger y-axis $\mathbf{y}_f^p$, and a preferred finger x-axis $\mathbf{x}_f^p$. We then require that $\theta_0 = 0$, which implies (by the kinematics in Appendix A) that the y-axis must be perpendicular to the desired normal $\mathbf{n}_f^d$. So $\mathbf{y}_f = \mathbf{y}_f^p - (\mathbf{y}_f^p \cdot \mathbf{n}_f^d)\mathbf{n}_f^d$, normalized to a unit vector. The x-axis must be perpendicular to this, so we project the preferred $\mathbf{x}_f^p$ onto the plane perpendicular to $\mathbf{y}_f$, giving $\mathbf{x}_f = \mathbf{x}_f^p - (\mathbf{x}_f^p \cdot \mathbf{y}_f)\mathbf{y}_f$, normalized to a unit vector. (It is possible that one of these projections will not exist, so "backup" preferences must be supplied for both $\mathbf{y}_f^p$ and $\mathbf{x}_f^p$.) Next we find the angle by which $\mathbf{x}_f$ must be rotated about $\mathbf{y}_f$ to reach the desired normal $\mathbf{n}_f^d$. This must be equal to the sum $\theta_{123} = \theta_1 + \theta_2 + \theta_3$ (by the kinematics in Appendix A). If the calculated angle lies outside the interval permitted by the sum of the joint angle limits, then $\mathbf{x}_f$ must be rotated about $\mathbf{y}_f$ to make the angle valid. Now $\mathbf{z}_f$ can be calculated by cross-product, and the rotation of the finger frame is fully determined. Next we specify the joint angles. We require that $\theta_0 = 0$, and that the other three joints be related by fixed ratios. Forward kinematics then gives the position of the fingertip in the local finger frame. Since its desired position in the world frame, $\mathbf{p}_f^d$, was a given, we can calculate the world position of the finger frame origin. The fixed transform from the finger to hand frame then yields the fully-determined hand coordinate frame.

Finally we test the hand transform to see if it is reachable by the arm kinematics. If not, we apply the arm kinematics to generate a nearby hand transform which is reachable. From this we calculate our 6 seed parameters, $(\mathbf{q}, \mathbf{t}) = (q_x, q_y, q_z, t_x, t_y, t_z)$.

Our hand-arm system offers 10 degrees of freedom: 4 finger joint angles and 6 arm angles. On these, we have imposed 5 constraints: preference on $\mathbf{y}_f$; preference on $\mathbf{x}_f$; $\theta_0 = 0$; $\theta_2 = k_2\theta_1$; $\theta_3 = k_3\theta_1$. We were given as inputs 6 values for the single finger: 3 position ($\mathbf{p}_f^d$) and 3 normal ($\mathbf{n}_f^d$). There is one constraint on these ($\mathbf{n}_f^d \cdot \mathbf{n}_f^d = 1$), leaving 5 independent input variables. So the degrees of freedom agree: $10 - 5 = 6 - 1$.

5.2 Combining several single-finger seeds

The justifications for computing the single-finger seeds above are that: (1) The computations are straightforward; (2) If a hand transform is known to work well for one finger, it's worth trying to see if it can accommodate the other fingers, too; (3) A composite of single-finger transforms might turn out to be a good seed.

To exploit the third advantage, we want to find a transform which is as similar as possible to all the single-finger seeds. First we must specify how to measure "similar" or "different" when dealing with two transforms. For the translation component, it seems clear: Euclidean distance in R^3. But it is not so obvious how to handle the rotation component.

5.2.1 "Difference" between rotations

We want a measure of how similar or different two rotations are. Let us consider still a third rotation, which sends one of the two rotations into the other. That rotation has an axis s and an angle θ. Since we have no reason to prefer any one axis of rotation over another, s should not be relevant to our measure. So a reasonable measure of the difference between two rotations is the angle θ of the rotation needed to send one into the other.

With quaternions, it is easy to find this angle. Given two rotations, with corresponding unit quaternions $\bar{\mathbf{q}} = [q_0, \mathbf{q}]$ and $\bar{\mathbf{r}} = [r_0, \mathbf{r}]$, the rotation which sends one into the other is:

$$\bar{\mathbf{q}}\,\bar{\mathbf{r}}^{-1} = [q_0 r_0 + \mathbf{q}\cdot\mathbf{r},\quad r_0\mathbf{q} - q_0\mathbf{r} - \mathbf{q}\times\mathbf{r}]$$

The angle of this rotation is

$$\theta = 2\arccos|q_0 r_0 + \mathbf{q}\cdot\mathbf{r}| = 2\arccos|\bar{\mathbf{q}}\cdot\bar{\mathbf{r}}|$$

This is simply twice the arc length between $\bar{\mathbf{q}}$ and $\bar{\mathbf{r}}$ on the unit quaternion sphere. For optimization, it is often desirable to use the squared value. A simple approximation is

$$\theta^2 \simeq 8(1 - |\bar{\mathbf{q}}\cdot\bar{\mathbf{r}}|)$$

derived from $|\bar{\mathbf{q}}\cdot\bar{\mathbf{r}}| = \cos(\theta/2) \simeq 1 - \theta^2/8$.

5.2.2 Fitting the optimal rotation

Our objective is to find the optimal fit to the set of given rotations (from the single-finger seed transforms). First get the quaternion form $\bar{\mathbf{q}}_i$ for the i-th seed rotation. (Take the $\mathbf{q}$ component from the seed parameters, and complete the full 4-component quaternion representation by $q_0 = (1 - \mathbf{q} \cdot \mathbf{q})^{(1/2)}$.)

Now using the approximation from the previous section, we want to find the quaternion $\bar{\mathbf{r}}$ which minimizes

$$\sum(1 - \bar{\mathbf{r}} \cdot \bar{\mathbf{q}}_i) = \sum 1 - \sqrt{1 - r_x^2 - r_y^2 - r_z^2}\sum q_{0i} - r_x \sum q_{xi} - r_y \sum q_{yi} - r_z \sum q_{zi}$$

This assumes that the signs of the seed quaternions have been adjusted so that they all point in more or less the same direction in R^4 (i.e. $\bar{\mathbf{q}}_i \cdot \bar{\mathbf{q}}_j \geq 0$), so that we can omit the absolute value operator. Differentiating with respect to r_x and setting the result to 0 yields

$$\frac{r_x}{r_0} = \frac{\sum q_{xi}}{\sum q_{0i}}$$

with corresponding results for r_y and r_z. Some more algebra gives

$$r_0 = \frac{\sum q_{0i}}{\left[(\sum q_{0i})^2 + (\sum q_{xi})^2 + (\sum q_{yi})^2 + (\sum q_{zi})^2\right]^{\frac{1}{2}}}$$

Put simply, we find the optimal rotation by first adjusting signs so that the given quaternions all point in roughly the same direction, then adding them all up, and finally normalizing the result to a unit quaternion $\bar{\mathbf{r}}$.

5.2.3 Completing the new seed parameters

Having computed this optimal quaternion $\bar{\mathbf{r}}$, we adjust the sign to make $r_0 \geq 0$, and take the (r_x, r_y, r_z) components as the rotational portion of the new optimal transform.

To get the optimal translation component, we take the arithmetic mean of the t vectors from the given seeds (a standard result of linear least squares optimization).

So we have a measure of the difference between two transforms:

$$\text{distance}[(\bar{\mathbf{q}}_1, \mathbf{t}_1), (\bar{\mathbf{q}}_2, \mathbf{t}_2)]^2 = 8(1 - |\bar{\mathbf{q}}_1 \cdot \bar{\mathbf{q}}_2|) + w(\mathbf{t}_1 - \mathbf{t}_2) \cdot (\mathbf{t}_1 - \mathbf{t}_2)$$

where w gives the weighting between translational and rotational differences. And based on this difference measure, we have given a method for finding a transform which is an optimal fit to a set of given transforms.

5.3 Putting it all together

The algorithm in section 5.1 for generating a seed transform from a single finger location must be supplied with a preferred rotation, given as preferred $\mathbf{x}_h$ and $\mathbf{y}_h$ basis vectors for the hand frame. Our practice is to try several preferred rotations, chosen mainly because they are likely to be reachable by the arm kinematics. So we supply a set of rotations based on the arm coordinate frame. Actually, since the base of our arm is held fixed, we can simply use the world frame. We also try as a preference the $\mathbf{x}_h$ and $\mathbf{y}_h$ from the current hand transform. The idea is that the current hand transform was probably chosen to be relevant to the task we are about to perform, and is already known to be reachable by the arm. The set of preferred rotations is:

$\mathbf{x}_h = (1,0,0)$	$\mathbf{y}_h = (0,-1,0)$
$\mathbf{x}_h = (1,0,0)$	$\mathbf{y}_h = (0,0,-1)$
$\mathbf{x}_h = (0.92,-0.38,0)$	$\mathbf{y}_h = (0,-1,0)$
$\mathbf{x}_h = (0.92,-0.38,0)$	$\mathbf{y}_h = (0,0,-1)$
$\mathbf{x}_h = (0.7,-0.7,0)$	$\mathbf{y}_h = (0,-1,0)$
$\mathbf{x}_h = (0.7,-0.7,0)$	$\mathbf{y}_h = (0,0,-1)$
$\mathbf{x}_h =$ current $\mathbf{x}_h$	$\mathbf{y}_h =$ current $\mathbf{y}_h$

The third one is a hand transform in which the fingertip positions all lie roughly on a horizontal plane (with the finger joints in the middle of their angle ranges).

For each preferred rotation, we generate a seed from each single desired finger location $(\mathbf{p}_f^d, \mathbf{n}_f^d)$ (section 5.1). Then we find the optimal fit to all the single-finger seeds from that preferred rotation (section 5.2). As one additional seed, we also try the current hand transform as is, without any modification of rotation or translation. This handles the case where the desired move is a small adjustment. Then the seeds are sorted in increasing order of cost function value, and the lowest one is optimized.

5.4 Further ideas on generating seeds

There are many possible modifications and expansions of our seed-generating procedure:

- In addition to using the composite optimal fit to all the seeds from a preferred rotation, try optimal fits to each pair, and each triple.
- Try other ratios of $\theta_1 : \theta_2 : \theta_3$ and other values of θ_0 than the single options used in the algorithm above. Even use different ratios and values for different fingers.
- Supply preferred rotations, and joint angle ratios and values, which are based on specialized grasp configurations ("power grasp", "finger-thumb pinch", "hook grasp", etc).

- Bypass the single-finger-and-then-combine procedure above, and use special calculations for special cases, such as all positions roughly in a single plane and all desired normals roughly parallel to that plane.
- Subject every seed to one iteration of the optimization algorithm before choosing the best for full optimization. The first iteration can often yield a large reduction in the cost function.

The first is easy, but so far we have gotten along fine without it. The next three are perhaps better carried out in conjunction with the design of grasp-planning and task-planning modules for the hand-arm system. The last can be settled only by more experiments. The point is that we have demonstrated a framework which is easy to extend. If the hand-arm coordination module is having trouble finding good grasps for a particular task domain, simply install some new seed-generating algorithms.

6 Results

We have run our hand-arm coordination procedure on a variety of different configurations of finger locations. Each test run was conducted as follows: We have our program read a file which contains the current hand transform parameters and the desired fingertip position and surface normal vectors, $\mathbf{p}_f^d$ and $\mathbf{n}_f^d$, together with their associated weights, w_{Pf} and w_{Nf}. The program prints out the generated seeds, optimizes the one with the lowest cost function, prints out the resulting hand transform, finger joint and arm angles, and the resulting fingertip position and surface normals. A representative run is shown in Figure 4. Diagrams of some of the configurations of desired locations that we did test runs on are shown in Figures 5 and 6. The tip of each arrow is the desired position, the direction is the desired normal, and the number is the number of the finger assigned to reach that location.

All the runs resulted in success, except some which were deliberately constructed to be unreachable. So we have evidence that this hand-arm coordination algorithm indeed works. With 2-finger configurations, the position is very accurate, with very small differences in the normals. With 3- and 4-finger configurations, the positions are usually within 0.2 cm (though once it was nearly 1 cm), and the normals get a little further off, as in the run in Figure 4. Part of the reason for the error on the normals is that it is kinematically impossible to attain exactly the desired normals with the Utah/MIT hand. We chose the sets of locations to simulate the output of a grasp stability planner, not to be exactly convenient for our particular mechanism. Of course, even if a run results in failure, that most likely shows only that we need to find better heuristics for generating seeds — an area where we readily acknowledge a sizable space for improvement.

We also did some runs with our Utah/MIT robot hand-arm system, which is described in [Allen *et al.*, 1989]. The above procedure was followed, except that the initial hand transform is derived by querying the PUMA 560 position sensors, and that the PUMA 560 is actually commanded to move to the desired transform, and the Utah/MIT hand commanded to move to the desired joint angles. Figure 7 shows two photographs of the hand applying a 4-finger grasp to a coffee mug.

Finally, we tried to actually use the configuration of desired locations to grasp an object. We tried the two different grasps on a coffee mug of Figure 5, and the two on a circular roll of tape of Figure 6. Photographs of two successes are shown in Figure 8. The procedure was as follows: Calculate the arm transform (the "grasp position" and "grasp orientation") and finger joint angles ("grasp angles") as before. Then use the arm to move the hand to a position 10cm in the $-z_h$ direction from its grasp position (z_h points outward from the palm) and in the grasp rotation. On each finger which is active in the grasp, move joint 0 to its grasp angle, and move each of joints 1, 2, and 3 to an angle 0.15 radians less than its grasp angle (which has the effect of opening the fingers wider apart). Use the arm to move the hand to its grasp position and rotation. Move joints 1, 2, and 3 to their grasp angles (which closes them to contact with the object, or nearly so). Then move each fingertip 1 cm in the direction of the normal n_f. This command cannot succeed, since the fingertips would then penetrate the object surface. But the Utah/MIT analog position controller will as a result command a force in the direction of the desired position, and that force is usually sufficient to hold the object. The autonomous procedure ends here, and we manually move the arm and to lift the hand and see if the object falls out or not.

Of course, this did not always work. Since no path planning or collision avoidance was done, sometimes a finger bumped the cup's handle aside during approach. Without using tactile or force sensing to verify and improve the quality of contact, a finger sometimes slipped off the cup handle.

7 Conclusion

We have presented a procedure for coordinating the positioning of a multi-finger hand-arm system, and used it to actually command a robot manipulator. We believe that the approach and many of the specific techniques from this research prototype can be taken up into a larger system for planning and executing grasps and manipulation tasks.

We are glad to acknowledge the encouragement of our advisor Peter Allen in pursuing this problem, and helpful conversations with Terry Boult and Kicha Ganapathy.

Appendix A: Kinematics of the fingers, hand, arm

Coordinate frames

There are three important coordinate frames for our hand-arm system: world (w), hand (h), and finger (f). Our world frame is the standard frame of the PUMA 560. The current actual hand frame is calculated from the PUMA 560 kinematics concatenated with the fixed transform due to the end effector connection to our Utah/MIT hand. Figure 2 shows how the finger frames are related to the hand coordinate frame. The hand frame is identical to the frame for finger 0. The transform from the hand frame to a local frame for finger 1, 2, or 3 is a translation, followed by a rotation about the hand y-axis[3]. There is also an arm frame, but since the base of our arm (the Utah/MIT remotizer) is fixed, this is essentially the same as the world frame.

Finger kinematics

Our Utah/MIT robot hand has 4 fingers, each finger has 4 joints, and θ_{fj} is the angle of joint j on finger f (see Figure 1). The forward kinematics for the position $\mathbf{p}^f = (p_x^f, p_y^f, p_z^f)$ (superscript f denotes local finger frame) and outward normal $\mathbf{n}^f$ of the pad of the distal link of the finger (see Figure 1) is

$$\begin{aligned} p_x^f &= L_{0x} + L_1 \sin\theta_1 + L_2 \sin\theta_{12} + L_{3w} \sin\theta_{123} + L_{3x} \cos\theta_{123} \\ p_y^f &= (L_{0w} + L_1 \cos\theta_1 + L_2 \cos\theta_{12} + L_{3w} \cos\theta_{123} - L_{3x} \sin\theta_{123}) \sin\theta_0 \\ p_z^f &= (L_{0w} + L_1 \cos\theta_1 + L_2 \cos\theta_{12} + L_{3w} \cos\theta_{123} - L_{3x} \sin\theta_{123}) \cos\theta_0 \end{aligned}$$

$$\mathbf{n}^f = (\cos\theta_{123}, -\sin\theta_{123}\sin\theta_0, -\sin\theta_{123}\cos\theta_0)$$

where $\theta_{12} = \theta_1 + \theta_2$ and $\theta_{123} = \theta_1 + \theta_2 + \theta_3$. The inverse kinematics can be solved in closed form, but will not be given here.

Arm kinematics

The arm for our hand is the Utah/MIT remotizer unit, supplied with the hand. The remotizer connects the hand to the pneumatic actuator pack, and bears the tendons which actuate the fingers. The remotizer itself is passive, so the position and orientation of the whole hand must be actuated externally. We use a PUMA 560 robot to provide 6 degrees of freedom in actuation at the base of the hand.

[3]Because of the sense in which θ_0 is measured by the Utah/MIT hand sensors, it is convenient to make the local frame for fingers 1, 2, and 3 left-handed. We ignore this in our presentation here (but account for it in our computations).

The presence of two "arm-like" linkages attached to the Utah/MIT hand is due to it being a research device. We may assume that future robot hands will have finger and hand actuation unified in a single arm. For the coordination problem in this paper, we have decided to ignore the PUMA 560, and only consider the remotizer as our single arm. (We are not concerned with dynamics, and the kinematic restrictions arise mainly from the remotizer).

The arm linkage and associated coordinate frames are shown in Figure 3. We model it as having 6 degrees of freedom, though actually there is a seventh, the twisting of the "upper arm" link. The forward and inverse kinematic formulas are straightforward. (There is a double solution in the inverse kinematics, but this is easily resolved for our purposes, by imposing a convention).

References

Acton, F. S. 1970. *Numerical Methods That Work*, Harper and Row, New York.

Allen, P. K., P. Michelman, and K. S. Roberts. 1989. "An Integrated System for Dextrous Manipulation," *Intl Conf Robotics Automation.*

Bay, J. S. 1989. "Tactile Shape Sensing via Single- and Multi-Fingered Hands," *Intl Conf Robotics Automation*, pp. 290–295.

Chou, J. C. K. and M. Kamel. 1988. "Quaternions Approach to Solve the Kinematic Equation of Rotation of a Sensor-Mounted Robotic Manipulator," *Intl Conf Robotics Automation*, pp. 656–662.

Funda, J. and R. P. Paul. 1988. "A Comparison of Transforms and Quaternions in Robotics," *Intl Conf Robotics Automation*, pp. 886–891.

Gill, P. E., W. Murray, and M. H. Wright. 1981. *Practical Optimization*, Academic, New York.

Goldstein, H. 1980. *Classical Mechanics (2nd edition)*, Addison-Wesley, Reading, MA.

Grupen, R. A. and T. C. Henderson. 1988. "A Control Paradigm for General Purpose Manipulation Systems," *Intl Conf Robotics Automation*, pp. 710–715.

Gu, Y. 1988. "Analysis of Orientation Representations by Lie Algebra in Robotics," *Intl Conf Robotics Automation*, pp. 874–879.

Hopf, H. 1940. *Systeme symmetrischer Bilinearformen und euklidische Modelle der projektiven Raume*, Vierteljschr. Naturforsch. Ges. Zurich.

Horn, B. P. K. May 1987. "Closed-form Solution of Absolute Orientation using Unit Quaternions," *Journal of the Optical Society A*, vol. 4, no. 4, pp. 629–642.

Li, Z. and S. S. Sastry. 1988. "Task-Oriented Optimal Grasping by Multifingered Robot Hands," *J Robotics Automation*, vol. RA-4(1), pp. 32–44.

Li, Z., P. Hsu, and S. S. Sastry. 1989. "Grasping and Coordinated Manipulation by a Multifingered Robot Hand," *Intl J Robotics Res*, vol. 8, no. 4, pp. 33–50.

Lin, S. 1988. "Euler Parameters in Robot Cartesian Control," *Intl Conf Robotics Automation*, pp. 1676–1681.

Lozano-Perez, T. *et al.* 1987. "Handey: A Robot System that Recognizes, Plans, and Manipulates," *Intl Conf Robotics Automation*, pp. 843–849.

Nguyen, V. 1988. "Constructing Force-Closure Grasps," *Intl J Robotics Res*, vol. 7, no. 3, pp. 3–16.

Nguyen, V. 1989. "Grasping and Coordinated Manipulation by a Multifingered Robot Hand," *Intl J Robotics Res*, vol. 8, no. 1, pp. 26–37.

Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. 1988. *Numerical Recipes in C*, Cambridge U. P., New York.

Rao, K., G. Medioni, H. Liu, and G. A. Bekey. 1988. "Robot Hand-Eye Coordination: Shape Description and Grasping," *Intl Conf Robotics Automation*, pp. 407–411.

Roberts, K. S., G. Bishop, and S. K. Ganapathy. 1988. "Smooth Interpolation of Rotational Motions," *IEEE Computer Vision and Pattern Recognition.*

Shoemake, K. 1987. "Quaternion Calculus and Fast Animation, in Course Notes # 10: Computer Animation: 3-D Motion Specification and Control," *ACM Computer Graphics.*

Spring, K. W. 1986. "Euler Parameters and the Use of Quaternion Algebra in the Manipulation of Finite Rotations: A Review," *Mechanism and Machine Theory*, vol. 21, no. 5, pp. 365–373.

Stansfield, S. A. 1988. "Reasoning about grasping," *AAAI*, pp. 768–773.

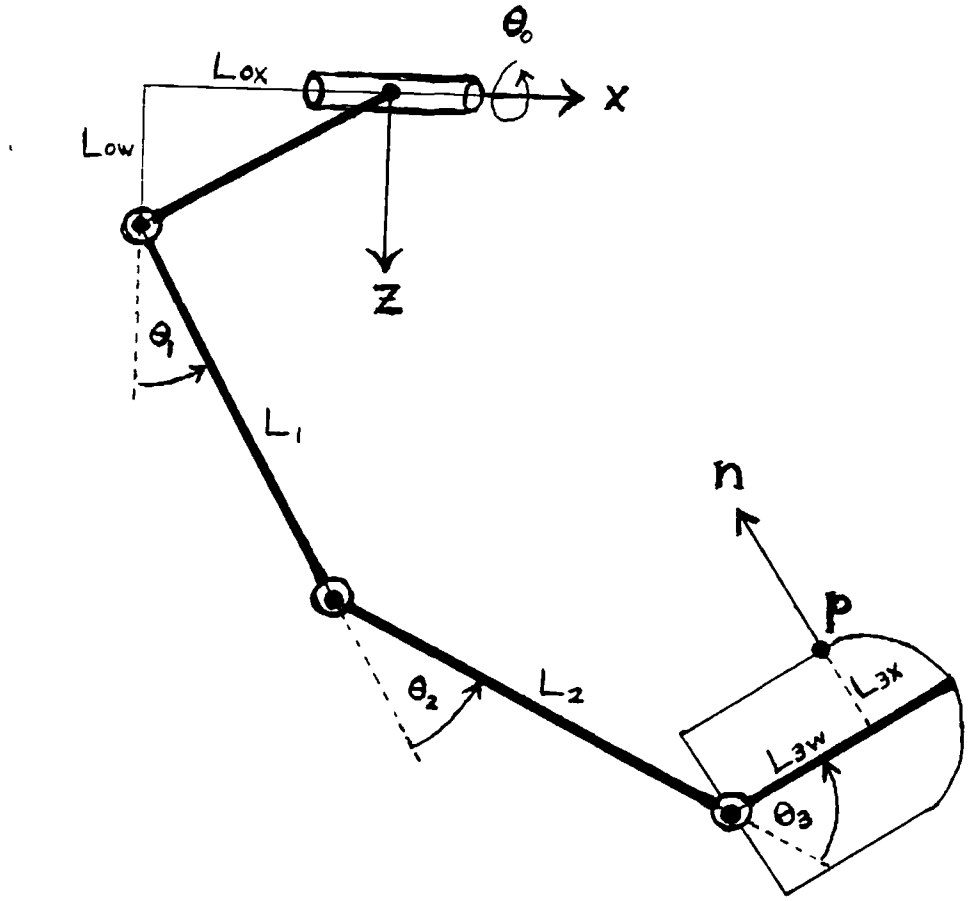
Stansfield, S. A. 1989. "Robotic Grasping of Unknown Objects: A Knowledge-Based Approach," *(to appear).*

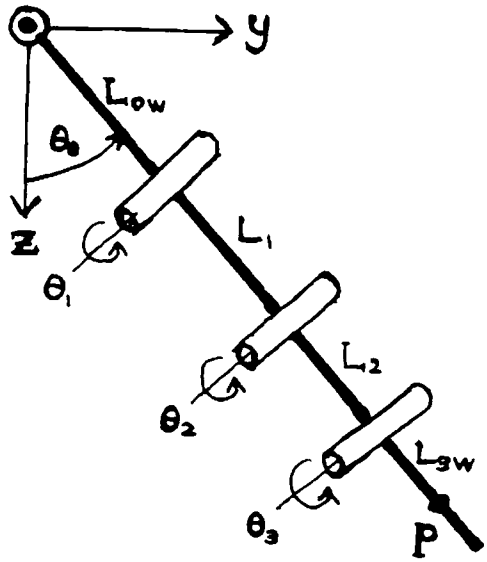
Stuelpnagel, J. 1964. "On the Parametrization of the Three-Dimensional Rotation Group," *SIAM Review*, vol. 6, no. 4, pp. 422–430.

Sugano, S. and I. Kato. 1987. "WABOT-2: Autonomous Robot with Dexterous Finger-Arm Coordination Control in Keyboard Performance," *Intl Conf Robotics Automation*, pp. 90–97.

Tomovic, R., G. Bekey, and W. Karplus. 1987. "A strategy for grasp synthesis with multifingered robot hands," *Intl Conf Robotics Automation*, pp. 83–89.

Figure 1. Finger kinematics





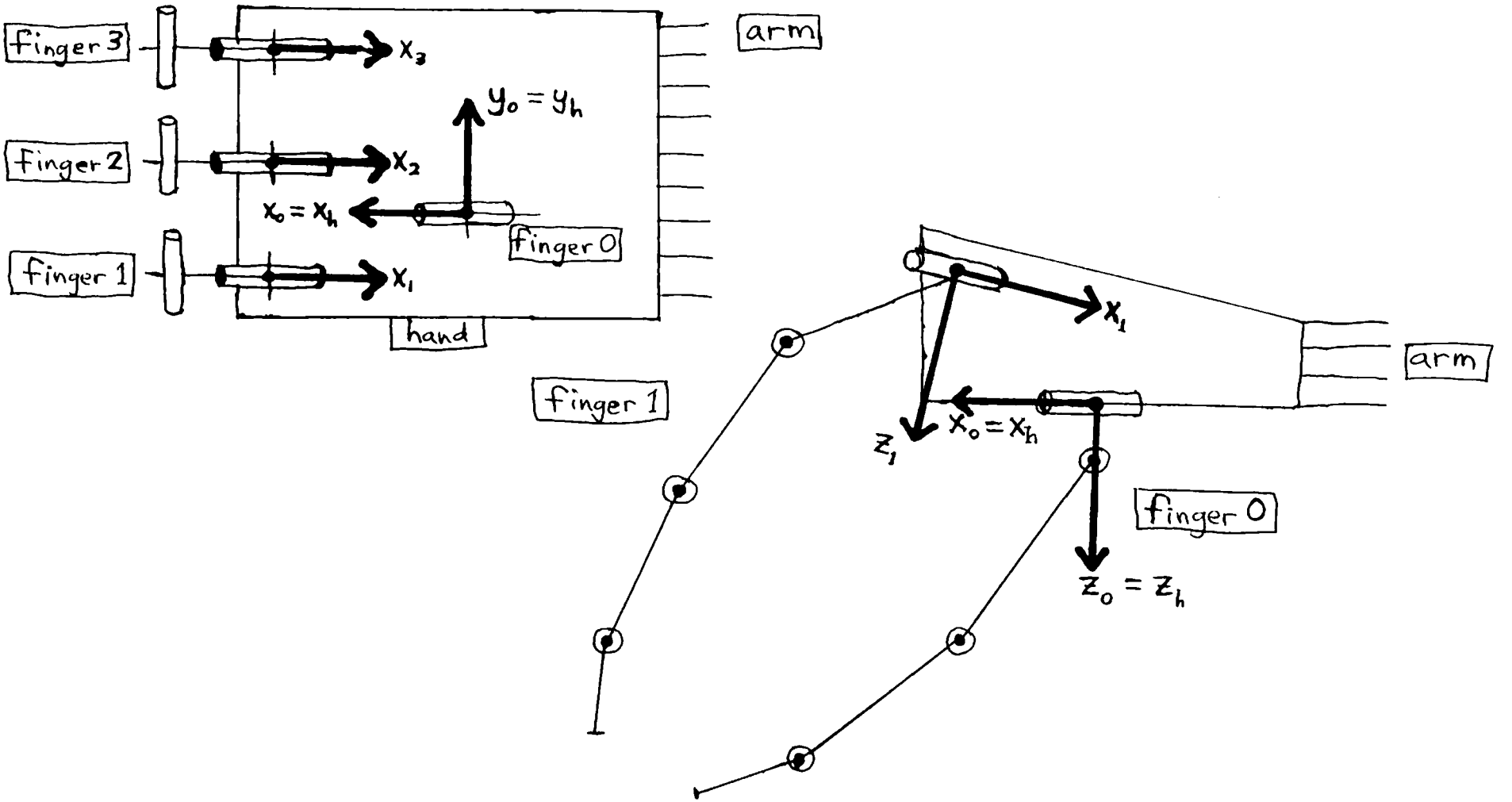


Figure 2: Hand and finger frames

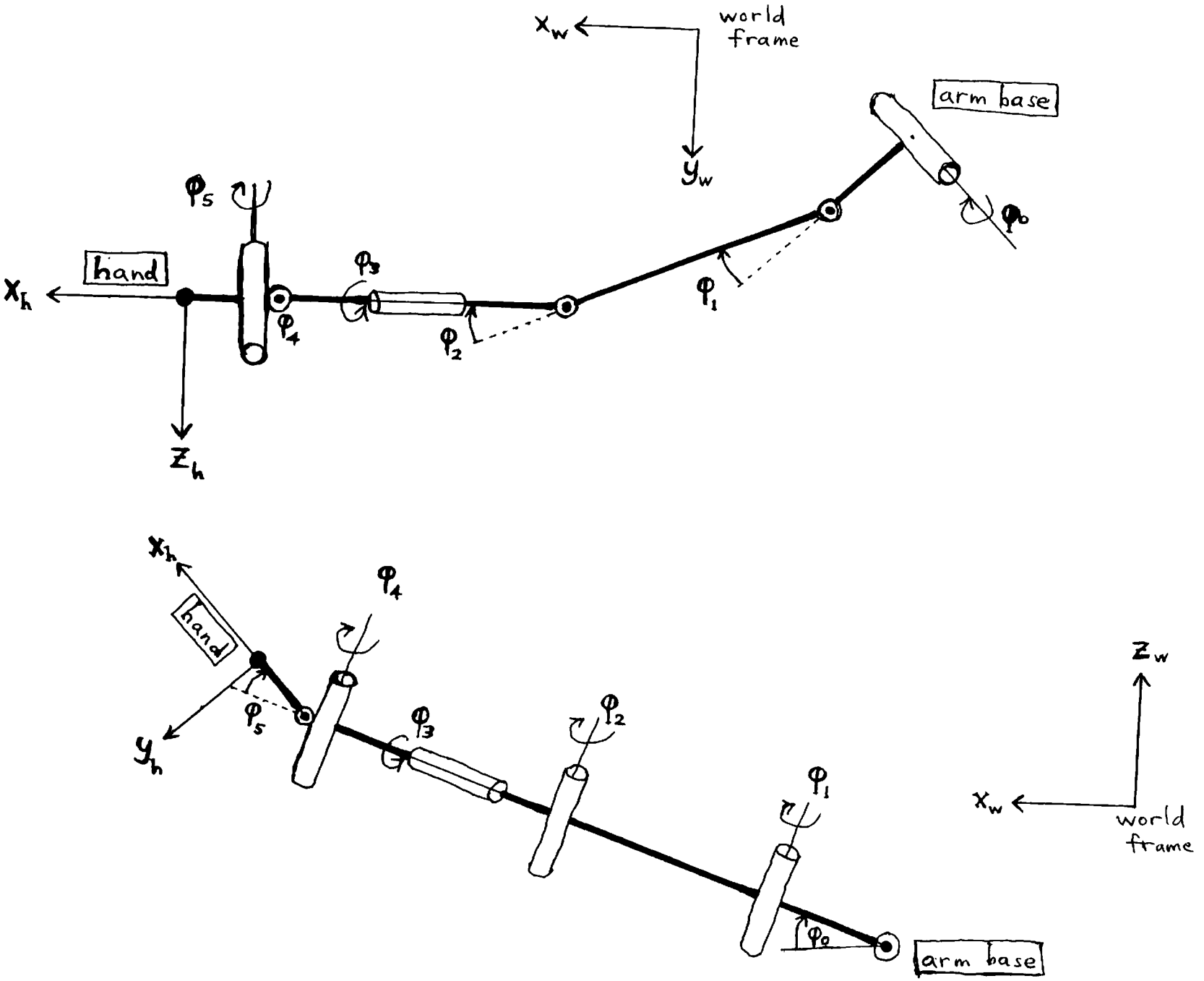
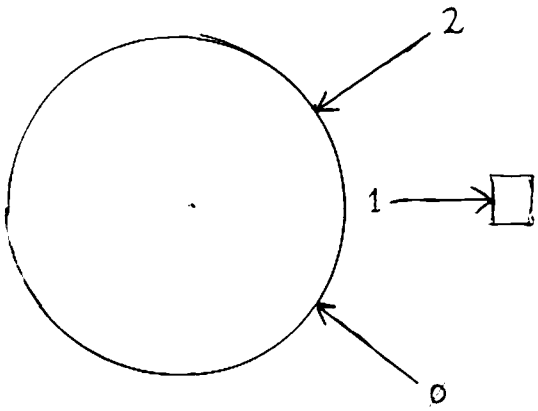


Figure 3. Arm kinematics

```
Desired locations:
f     position Pdf                normal Ndf
0:   -63.43   9.02 -37.98        0.88  0.14  0.45
1:   -60.04  11.72 -35.51       -0.10  0.15  0.98
2:   -58.38   8.46 -37.87       -0.80  0.08  0.59

Weights:
finger =       0          1          2          3
Wpf =       3.0000     3.0000     3.0000     0.0000
Wnf =       3.0000     3.0000     3.0000     0.0000

List of seed transforms:
cost          Qx       Qy       Qz       Tx       Ty       Tz
21.0533 : -0.9982 -0.0367  0.0364 -65.1917  9.3853 -28.5435
25.6809 : -0.9993  0.0304 -0.0007 -64.8820 11.0792 -31.9166
31.8367 : -0.9989 -0.0354  0.0011 -65.5392  9.3426 -28.5464
50.0031 : -0.9970  0.0077  0.0007 -64.0076 10.6667 -30.6129
58.2877 :  0.8480  0.0199 -0.1189 -63.6266  1.2648 -30.6599
64.0419 : -0.9943  0.0019 -0.0734 -64.2873 10.5282 -31.5359
76.4765 :  0.7574  0.0597 -0.1407 -64.0987  0.3806 -30.5104
78.7204 : -0.9958 -0.0200 -0.0851 -68.3167  8.9629 -29.1910
79.8557 :  0.9323 -0.0066 -0.2698 -62.9870  4.0595 -26.2353
81.1432 :  0.5000  0.0593 -0.0585 -64.5073  5.3595 -35.7432
111.147 :  0.5014  0.0313 -0.0310 -65.5947  6.9040 -34.7874
141.772 : -0.9833 -0.0609  0.1570 -56.8624  9.6947 -28.2748
179.355 :  0.7029  0.0564 -0.1041 -64.8136 -0.2260 -31.6973
204.236 :  0.7772  0.0049  0.0645 -63.7941 -0.6456 -35.2340
262.293 :  0.5033  0.0646 -0.0588 -60.1771  2.0083 -34.2534
266.375 :  0.6587  0.0680  0.0288 -62.2701 -0.9646 -38.1590
454.956 :  0.6907  0.1284 -0.1926 -61.5513 -4.8594 -34.5647
484.384 : -0.9584 -0.0470  0.2777 -53.0521 10.6682 -25.0205
808.948 : -0.9471 -0.1125  0.2707 -49.2184  9.4529 -30.6128
1541.51 :  0.0586 -0.0359 -0.0127 -69.7005 21.9026 -34.5058
2203.61 :  0.8897  0.0911  0.1084 -69.0000 18.0000 -19.6000
2240.79 :  0.0714 -0.0206 -0.0141 -56.7099 11.8489 -30.0365

number of iterations = 5
result:
cost          Qx       Qy       Qz       Tx       Ty       Tz
1.467   :  0.9875  0.0696 -0.1261 -63.4723 10.1174 -28.9198

hand transform:
      0.9585      0.1535     -0.2402     52.3384
      0.1215     -0.9823     -0.1426     13.5287
     -0.2579      0.1075     -0.9602    -45.2230
      0.0000      0.0000      0.0000      1.0000

PUMA 560 location =
         x         y         z         o         a         t
    ( -644.467, 63.580, -164.772, -125.773, 78.034, -73.455 )

Finger joint angles (radians):
        joint = 0          1          2          3
finger 0      0.2705     -0.1890     0.5558     0.3335
finger 1      0.1897     -0.0702     0.2099     1.3221
finger 2      0.1514     -0.1588     0.0900     0.6609
finger 3      0.0000      0.0000     0.0000     0.0000

Arm angles (Utah/MIT remotizer) (radians):

          0      1      2      3      4      5
phi =   0.086 -0.144  0.920 -1.480 -0.105 -0.322

Result locations:
f     position Pf                 normal Nf
0:   -63.43   9.02 -37.98        0.87  0.22  0.44
1:   -60.04  11.72 -35.51       -0.09  0.04  1.00
2:   -58.41   8.46 -37.63       -0.81 -0.10  0.58
```

Figure 4. Run of the 3 finger coffee mug grasp.



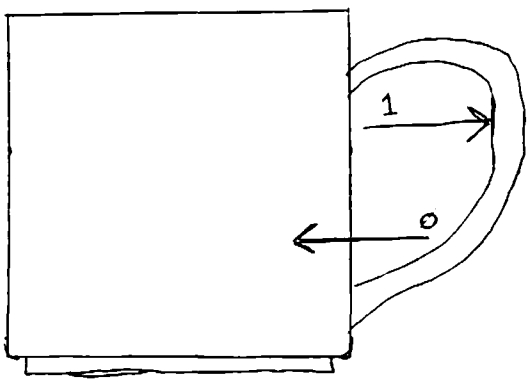
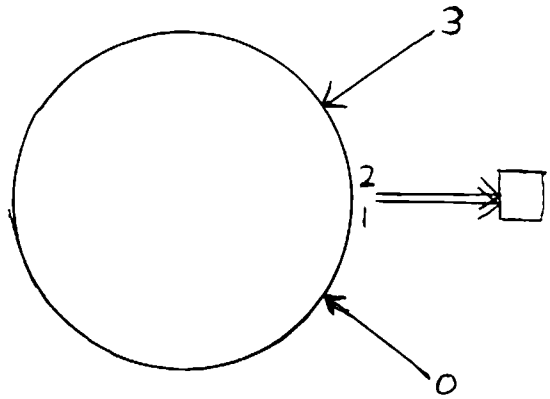


Figure 5a. 3-finger grasp of coffee mug



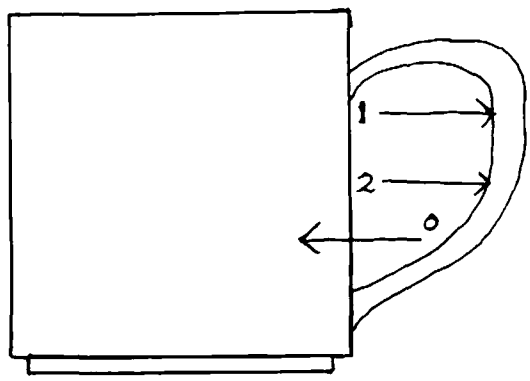


Figure 5b. 4-finger grasp of coffee mug

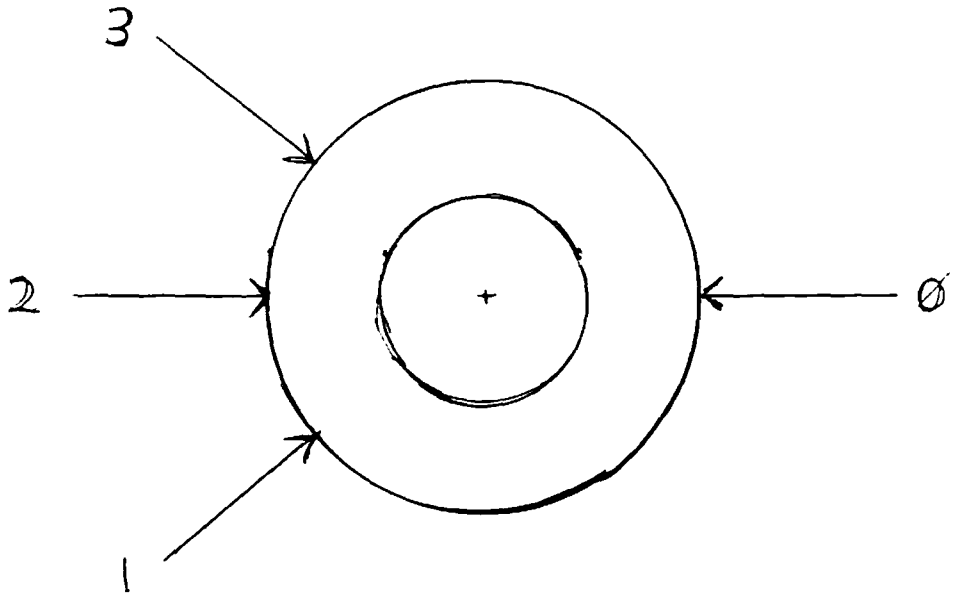


Figure 6a. 4-finger outside grasp of tape roll

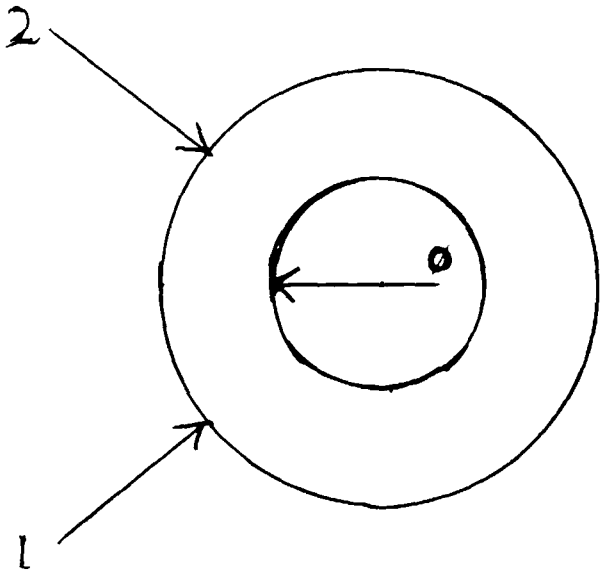


Figure 6b. 3-finger outside-inside grasp of tape roll.

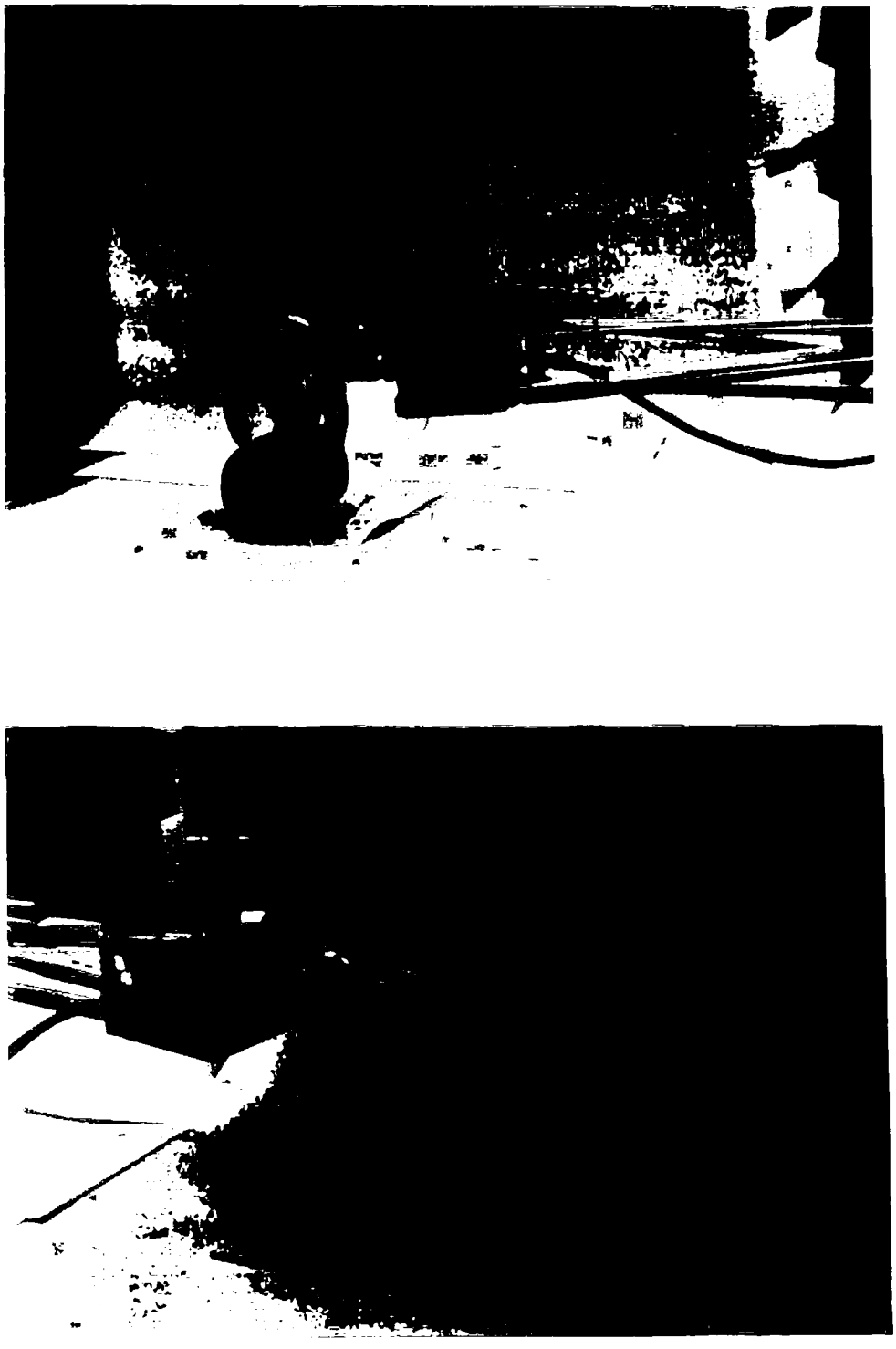


Figure 7: 4-finger grasp of coffee mug

Figure 8a: 3-finger grasp of coffee mug.

Figure 8b: 4-finger grasp of tape roll