ICPIK: Inverse Kinematics based Articulated-ICP

Shachar Fleishman* Mark Kliger* Alon Lerner Gershom Kutliroff
Intel
{shahar.fleishman,mark.kliger,alan.lerner,gershom.kutliroff}@intel.com

Abstract

In this paper we address the problem of matching a kinematic model of an articulated body to a point cloud obtained from a consumer grade 3D sensor. We present the ICPIK algorithm - an Articulated Iterative Closest Point algorithm based on a solution to the Inverse Kinematic problem. The main virtue of the presented algorithm is its computational efficiency, achieved by relying on inverse-kinematics framework for analytical derivation of the Jacobian matrix, and the enforcement of kinematic constraints. We demonstrate the performance of the ICPIK algorithm by integrating it into a real-time hand tracking system. The presented algorithm achieves similar accuracy as state of the art methods, while significantly reducing computation time.

1. Introduction

Recent advances in 3D imaging technology, notably Intel's RealSense [22] and Microsoft's Kinect [23] sensors, allow for 3D capture of the objects and people in a scene at high, interactive frame rates. The availability of this technology in a low-cost and small form factor package has generated much interest in the area of human-computer interaction, such as the problem of tracking a hand skeleton, which enables the design of interactive applications controlled by a user's natural movements.

Computing the skeleton of a hand based on data captured by a single camera is a challenging problem, due to viewpoint variability, the complex articulations of the fingers, and the prevalence of self occlusions caused by natural hand motions. This topic continues to be an active area of research in computer vision, even after many years. Earlier works focused primarily on input from RGB and grayscale images; for a comprehensive review, refer to Erol *et al*. [7]. The introduction of consumer grade 3D sensors has shifted the focus to methods based on the 3D data obtained by these devices. One possible approach is based on reconstructing a deformable surface model [5, 8, 11]. An alternative approach matches a hand model to the input depth image captured by the camera, which is done by solving an optimization problem [2, 12, 15, 17].

A natural representation for an articulated objects which possess an underlying skeletal structure, such as human hands and bodies, are kinematic chains of rigid bodies (bones) connected together by joints. The kinematics equations of the body define the relationship between the joint angles and its pose. The *forward kinematics* (FK) problem uses the kinematic equations to determine the pose given the joint angles and bones lengths. The *inverse kinematics* (IK) problem computes the joint angles for a desired pose of the articulated body.

In this work we present an efficient articulated iterative closest point algorithm for matching a kinematic model of an articulated body to a point cloud. The key idea is to solve the optimization step of ICP using an inverse kinematics solver. The solver is based on well-established techniques to compute analytic derivatives of the IK optimization function. This allows the efficient estimation of the non-rigid transformation of an articulated body in an ICP problem. Furthermore, it enables the enforcement of additional constraints which are non-standard in the ICP formulation, such as kinematic physical constraints, repulsive points that push the model away, and weighting methods. We therefore refer to our method as *ICPIK*.

The rest of this paper is organized as follows. Related works are presented in Section 2. Our optimization algorithm is described in Section 3. We have integrated our algorithm into a hand-tracking system which is briefly described in Section 4. We present results and conclusions in Section 5

2. Related work

An Iterative Closest Point (ICP) [1, 16] is an algorithm which finds the rigid transformation that aligns two point clouds. In each iteration, the algorithm updates the correspondence between the source and target point clouds, and computes the rigid transformation that best aligns them.

Multiple works have extended the ICP algorithm to han-

*S. Fleishman and M. Kliger contributed equally to this work.

dle non-rigid transformations. One approach is to reformulate the ICP problem as a non-linear optimization problem in the parameter space of the kinematic model of an articulated body [2, 6, 9, 14]. This frequently requires calculating the Jacobian matrix of partial derivatives of an optimization function. Dewaele *et al*. [6] apply the Levenberg-Marquardt (LM) method with the Jacobian matrix of the kinematic transformation, but the authors do not provide details regarding their implementation. Bray *et al*. [2] use a mix of computationally expensive numerical and algorithmic methods to compute the derivatives. Grest *et al*. [9] analytically derived the Jacobian matrix for human body pose estimation. We present a general, computationally efficient, solution to the non-rigid ICP problem based on the analytic derivatives of the kinematic model.

Our articulated ICP algorithm draws upon prior work on the IK problem by the robotics and computer graphics communities. In particular, the solution to the kinematic model benefits from an analytic formulation of the Jacobian matrix, as first derived by Orin and Schraeder [13] in 1984. The most similar approach to ours is the work of Grest *et al*. [9] who directly derive the Jacobian matrix for the particular case of a full human body. The authors employ a pseudoinverse-based solution to the optimization problem, which is known to be unstable in cases where the Jacobian is singular or nearly singular. To solve the inverse kinematics problem, we apply the Damped Least-Squares method, which is related to the LM method, and is known to be more stable for these type of problems [4]. Moreover, using an IK approach for solving an articulated ICP problem allows the seamless incorporation of kinematic joint constraints, joint weights, and point weights, as well as other constraints (see Section 3).

Articulated ICP methods are a core element of many full body and hand tracking systems. Typically, such systems are composed of two major components: (i) a segmentation and part detection module, usually based on machine learning algorithms [10, 17, 18, 21] or using visual markers [24, 25], which detects the approximate locations of the articulated body and its parts on an input frame , and (ii) a tracking system which computes deformations of a model of the body in order to match the input depth image, which is solved by some non-linear optimization. This work focuses on the latter problem. Oikonomidis *et al*. [12] use the Particle Swarm Optimization (PSO) algorithm with random initializations to search the parameter space and find the hand model configuration that best fits the data. Qian *et al*. [15] extends the approach of Oikonomidis *et al*., by incorporating an ICP step into the PSO algorithm. Specifically, at each PSO iteration, an additional ICP iteration is used in order to converge to faster a local minimum, where the ICP problem is solved with a partial LM optimization based on [14].

The proposed ICPIK algorithm can be incorporated as an optimization step in various full body and hand tracking systems, such as the ones described above. It is distinguished by its simplicity and computational efficiency. In Section 5, we compare the performance of the ICPIK and PSO algorithms integrated into the same tracking system, and demonstrate that ICPIK outperforms PSO in both speed and accuracy.

3. IK based articulated ICP: ICPIK

3.1. Inverse kinematics problem

An articulated body can be represented as a multi-body kinematic system consisting of a set of rigid objects, called *links* (bones), connected together by *joints*. Joints have a single degree of freedom, $DoF = 1$, and can be either rotational (revolute) or translational (prismatic). Other joint types, for example screw joints, can be represented by a combination of two or more of these basic joints connected by zero-length links. A rotational joint is parameterized by a rotation axis and a scalar angle value, while a translational joint is parameterized by a direction vector and translation distance. Note that the global 3D position and orientation of an articulated body can be represented by a root joint, which consists of three translational joints and three rotational joints, $DoF = 6$, i.e. 6 basic joints connected by zero-length links. An articulated body thus has n joints, each with $DoF = 1$, and an associated vector $\boldsymbol{\theta} = (\theta_1, ..., \theta_n)$, where θ_j is the kinematic parameter of the jth joint.

Certain points on the links, typically extremity points of kinematic chains, and the joints themselves, are identified as *end-effectors*. If there are k end-effectors, their 3D positions are denoted by $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_k)^T$. Each end-effector's position $\mathbf{s}_i$ is a function of $\boldsymbol{\theta}$, and is computed by applying the forward kinematic equations. The objective of the IK problem is to find the values of $\boldsymbol{\theta}$ that transform the joints so that the end-effectors $\mathbf{s}$ reach their target position. The target positions of the end-effectors are given by a vector $\mathbf{t} = (\mathbf{t}_1, \mathbf{t}_2, ..., \mathbf{t}_k)^T$. The IK problem can be stated as finding values of $\hat{\boldsymbol{\theta}}$ such that

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} ||\mathbf{t} - \mathbf{s}(\boldsymbol{\theta})||^2. \tag{1}$$

Equation (1) can be solved by using the Jacobian matrix to linearly approximate the function $\mathbf{s}(\boldsymbol{\theta})$. Recall that the Jacobian matrix of a vector valued function $\mathbf{s}(\boldsymbol{\theta})$ is the matrix of all first-order partial derivatives with respect to θ_i,

$$\mathbf{J}(\boldsymbol{\theta}) = \left(\frac{\partial \mathbf{s}_i}{\partial \theta_j} \right)_{i,j}. \tag{2}$$

In a simple kinematic models the Jacobian can be computed by manual differentiation. Alternatively, the Jacobian of

forward kinematics can be computed by symbolic or numerical auto-differentiation, which is often time consuming. We use the method of Orin and Scharader [13] to analytically calculate the entries in the Jacobian matrix for an *arbitrary* kinematic model. For the jth rotational joint with $DoF = 1$, let θ_j be its angle of rotation, $\mathbf{p}_j$ be its position, and let $\mathbf{v}_j$ be the unit vector pointing along its current axis of rotation. The corresponding entry in the Jacobian matrix for the rotational joint j affecting the ith end-effector is

$$\frac{\partial \mathbf{s}_i}{\partial \theta_j} = \mathbf{v}_j \times (\mathbf{s}_i - \mathbf{p}_j), \tag{3}$$

where the angles are measured in radians, and the direction of rotation is given by the right-hand rule. Intuitively, this equation means that an infinitesimal rotation around the axis $\mathbf{v}_j$ centered at $\mathbf{p}_j$ will move the end-effector $\mathbf{s}_i$ by an infinitesimal distance, proportional to distance between $\mathbf{s}_i$ and $\mathbf{p}_j$, along the direction defined by (3). If the ith end-effector is not affected by the jth joint, then $\frac{\partial \mathbf{s}_i}{\partial \theta_j} = 0$.

Similarly, for the jth translational joint with $DoF = 1$, let θ_j be its translation distance along its direction vector $\mathbf{v}_j$. If the ith end-effector is affected by the jth joint, then

$$\frac{\partial \mathbf{s}_i}{\partial \theta_j} = \mathbf{v}_j. \tag{4}$$

Let

$$\boldsymbol{\theta} := \boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}. \tag{5}$$

The end-effector positions can be linearly approximated by

$$\mathbf{s}(\boldsymbol{\theta}) \approx \mathbf{s}(\boldsymbol{\theta}_0) + \mathbf{J}(\boldsymbol{\theta}_0)\Delta\boldsymbol{\theta}. \tag{6}$$

For the sake of simplicity we omit the parameter vector $\boldsymbol{\theta}_0$ and denote the Jacobian matrix as $\mathbf{J}$. Using the linear approximation (6) we solve (1) by iteratively updating $\boldsymbol{\theta}$ from the previous iteration by $\Delta\boldsymbol{\theta}$ as obtained from

$$\arg\min_{\Delta\boldsymbol{\theta}} ||\mathbf{e} - \mathbf{J}\Delta\boldsymbol{\theta}||^2, \tag{7}$$

where the error vector $\mathbf{e}$ is defined as $\mathbf{e} := \mathbf{t} - \mathbf{s}(\boldsymbol{\theta}_0)$.

There are several methods to solve a least-squares problem such as (7) including SVD, the Jacobian transpose method, pseudoinverse, etc [3]. We use Damped Least Squares, also known as Levenberg-Marquardt optimization, which is numerically stable and fast. Rather than solving (7), we find the value of $\Delta\boldsymbol{\theta}$ that minimizes the l_2 regularized version of (7),

$$||\mathbf{e} - \mathbf{J}\Delta\boldsymbol{\theta}||^2 + \lambda||\Delta\boldsymbol{\theta}||^2 \tag{8}$$

where $\lambda > 0$ is the *damping constant*. Minimizing (8) with respect to $\Delta\boldsymbol{\theta}$ is equivalent, as shown in [3], to solving

$$\mathbf{J}^T\mathbf{e} = (\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I})\Delta\boldsymbol{\theta}. \tag{9}$$

The matrix on the right-hand side (RHS) of (9) is positive definite, and can therefore be solved efficiently using Cholesky factorization. Note that the number of equations in (9) is equal to the number of parameters n and is independent of the number of end-effectors m. Moreover, the matrix $\mathbf{J}^T\mathbf{J}$ and the vector $\mathbf{J}^T\mathbf{e}$ can be computed directly from (2) as follows:

$$\left(\mathbf{J}^T\mathbf{J}\right)_{jk} = \sum_{i=0}^{m} \frac{\partial \mathbf{s}_i}{\partial \theta_j} \cdot \frac{\partial \mathbf{s}_i}{\partial \theta_k}, \tag{10}$$

and

$$\left(\mathbf{J}^T\mathbf{e}\right)_j = \sum_{i=0}^{m} \frac{\partial \mathbf{s}_i}{\partial \theta_j} \cdot (\mathbf{t}_i - \mathbf{s}_i). \tag{11}$$

Substituting (3-4) into (10-11), we have

$$\left(\mathbf{J}^T\mathbf{J}\right)_{jk} = \sum_{i=0}^{m} \begin{cases} 0, j \text{ or } k \text{ are not connected to effector } i \\ (\mathbf{v_j} \times (\mathbf{s_i} - \mathbf{p_j})) \cdot (\mathbf{v_k} \times (\mathbf{s_i} - \mathbf{p_k})), \ j, k \text{ rot.} \\ (\mathbf{v_j} \times (\mathbf{s_i} - \mathbf{p_j})) \cdot \mathbf{v_k}, \ j \text{ rot., } k \text{ trans.} \\ \mathbf{v_j} \cdot \mathbf{v_k}, \ j, k \text{ trans.} \end{cases} \tag{12}$$

and

$$\left(\mathbf{J}^T\mathbf{e}\right)_j = \sum_{i=0}^{m} \begin{cases} 0, j \text{ is not connected to effector } i \\ (\mathbf{v_j} \times (\mathbf{s_i} - \mathbf{p_j})) \cdot (\mathbf{t_i} - \mathbf{s_i}), \ j \text{ rot.} \\ \mathbf{v_j} \cdot (\mathbf{t_i} - \mathbf{s_i}), \ j \text{ trans.} \end{cases} \tag{13}$$

Moreover, from (12) and (13) we can see that adding pairs of end-effectors and targets to the IK problem does not significantly increase the amount of computation. Indeed, the pair of end-effector $\mathbf{s}_i$ and target $\mathbf{t}_i$ only affects those entries of the Jacobian matrix $\left(\mathbf{J}^T\mathbf{J}\right)_{jk}$, where both the joints j and k, as well as the end-effector $\mathbf{s}_i$ belong to the same kinematic chain. Similarly, $\mathbf{s}_i$ and $\mathbf{t}_i$ only affect the entries of $\left(\mathbf{J}^T\mathbf{e}\right)_j$ in which both the joint j and end-effector $\mathbf{s}_i$ belong to the same kinematic chain.

Applying joint weights allows certain joints to move or rotate more easily than others. For example, weights $w_i > 0$ can be set to be proportional to the cost of changing the joint's parameter θ_i. In this case higher weight means the cost to change θ_i is higher relative to joints with low weights. Therefore, we reparametrize $\tilde{\theta}_i = w_i\theta_i$, and solve (9) for $\Delta\tilde{\boldsymbol{\theta}}$. Target weights $\nu_i > 0$ adjust the relative importance of targets t_i by multiplying the error vector e_i by the weight. Note that target weights do not affect the RHS of (9). Thus, we can reformulate (10) and (11) as

$$\left(\tilde{\mathbf{J}}^T\tilde{\mathbf{J}}\right)_{jk} = \sum_{i=0}^{m} \frac{1}{w_j w_k} \frac{\partial \mathbf{s}_i}{\partial \theta_j} \cdot \frac{\partial \mathbf{s}_i}{\partial \theta_k} \tag{14}$$

and

$$\left(\tilde{\mathbf{J}}^T\mathbf{e}\right)_j = \sum_{i=0}^{m} \frac{\nu_i}{w_j} \frac{\partial \mathbf{s}_i}{\partial \theta_j} \cdot (\mathbf{t}_i - \mathbf{s}_i) \tag{15}$$

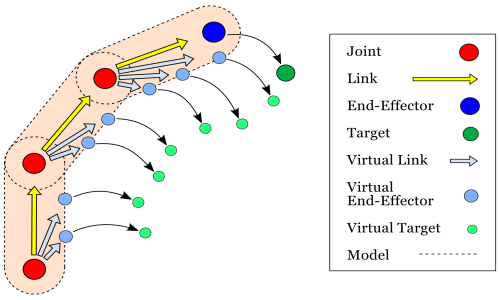


Figure 1. Target points (Green) are assigned to end-effectors (Blue). We add virtual end-effectors for targets that are not associated with any specific joint of the model by finding the closest point on the model to the target.

After calculating $\Delta\tilde{\boldsymbol{\theta}}$ we update $\boldsymbol{\theta}$ by $\Delta\theta_i = \Delta\tilde{\theta}_i / w_i$.

In addition, it is possible to define *repulsive targets* that push end-effectors away. Repulsive targets can, for example, prevent self intersections, move the model behind a visible surface, or move it away from regions in space that the body cannot occupy. These can be implemented by adding targets with *negative weights* which are inversely proportional to the distance from the end-effector, and updating the LHS of (9). For example, a repulsive spherical target $\mathbf{t}_i$ with radius r for the ith end-effector $\mathbf{s}_i$, can be defined by the weight

$$\nu_i = \min\left(-\frac{r - ||\mathbf{s}_i - \mathbf{t}_i||}{||\mathbf{s}_i - \mathbf{t}_i||}, 0\right). \tag{16}$$

The negativity of the weight changes the direction of the error vector $\mathbf{e}_i = \mathbf{t}_i - \mathbf{s}_i$.

Often joints in a kinematic model should obey restriction constraints. For example, finger's abduction/adduction and flexion/extension angles are restricted by physical limitations. This can be expressed in the kinematic solution by reformulating the problem as a bounded constraint optimization problem, in which each joint has a lower and upper limit, that is $\theta_i \in [l_i, h_i]$ which is its *feasible set*. We apply an *active set method* for constrained optimization in the following manner: when a constraint is inactive, that is, parameter value is within its feasible set, we apply the non-constrained optimization to it. When a constraint becomes active, θ_i is set to its closest limit, and in the following iteration, θ_i remains constant.

3.2. Combining ICP with IK

As previously mentioned, to formulate an IK problem certain points on the links have to be defined as end-effectors. Typically extremity points of kinematic chains, and the joints themselves, are defined as end-effectors. The set of end-effectors, their target positions and the initial value of the kinematic parameters define the IK problem.

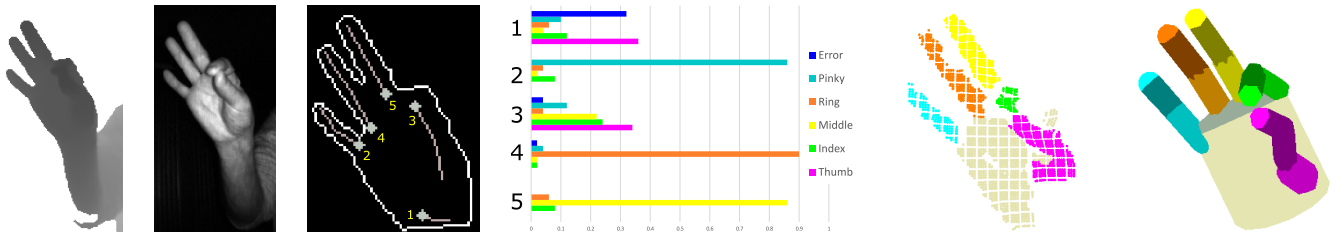
We are interested in generalizing the standard formulation by adding additional *(end-effector, target)* pairs, not necessarily lying on the links, to our IK problem. Specifically, we wish to work with an articulated body model, composed of a skeleton and an associated *skin*. In this case, points on the skin, a fixed distance away from any link, constitute our end-effectors. We refer to these points as *virtual end-effectors*, and their associated targets as *virtual targets*.

The process of choosing virtual end-effectors and targets is task dependent. Recall, that our goal is to estimate the pose of an articulated body to match the depth image. Thus, we define virtual end-effectors and targets in a manner similar to the ICP algorithm, i.e. we choose random points on the depth image as our virtual targets, and then designate the closest points on the model's skin as their associated virtual end-effectors. While the virtual end-effectors do not lie on any link, they can be associated with a parent joint. We assign a *virtual link* between each virtual end-effector and its associated parent joint by the vector originating at the parent joint and terminating at the end-effector, see Figure 1. Pairs of virtual end-effectors and their targets participate in the formulation of the IK problem simply by increasing the size of the summation in (12) and (13) by the number of virtual targets. It should be noted that adding pairs of virtual end-effectors and their targets does not increase the sizes of the matrix $\mathbf{J}^T\mathbf{J}$ and the vector $\mathbf{J}^T\mathbf{e}$. Therefore, the additional computational costs are small.

Finally, in contrast to the standard IK formulation, in which the targets remain constant throughout all the iterations of the IK solver, pairs of virtual end-effectors and their targets are updated at every iteration. Thus, our algorithm is iterative, and similar in spirit to the standard ICP approach which computes correspondences between two sets of points at each iteration. However, in contrast to ICP, we calculate the transformation between the point sets with an IK solver, thus generating a non-rigid transformation of the articulated body.

Below is the pseudocode of the ICPIK algorithm. M is the model, $\mathbf{p}$ are the joints positions, and $\mathbf{v}$ are their axes:

$(M, \mathbf{p}, \mathbf{v}) = ForwardKinematics(\boldsymbol{\theta}_0)$;
for $i = 1$ to L
  $\mathbf{t} = SampleTargets(D)$;
  $\mathbf{s} = FindCorrespondence(\mathbf{t}, M)$;
  Compute $\mathbf{J}^T\mathbf{J}$, $\mathbf{J}^T\mathbf{e}$;
  Find $\Delta\boldsymbol{\theta}$ from (8);
  $\boldsymbol{\theta}_i = \boldsymbol{\theta}_{i-1} + \Delta\boldsymbol{\theta}$;
  $(M, \mathbf{p}, \mathbf{v}) = ForwardKinematics(\boldsymbol{\theta}_i)$;
end



(a) (b) (c) (d) (e) (f)

Figure 2. Data flow overview: (a) depth image, (b) grayscale image, (c) candidate fingers, (d) finger label probabilities, (e) labeled targets on the depth image from which we subsample, (f) output hand model

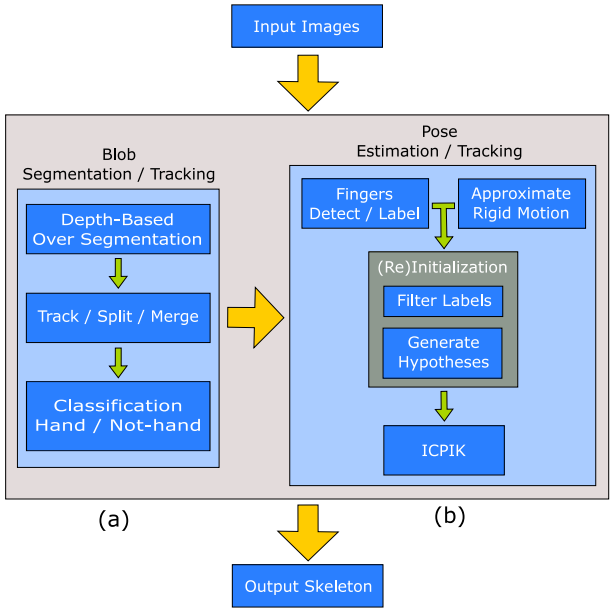


Figure 3. The proposed system has two main components: (a) a blob segmentation and tracking component; (b) a pose estimation and tracking component which estimates an initial position for the hand and then apply the ICPIK algorithm.

4. Real-time hand pose tracking

In order to provide a context for the ICPIK algorithm, we present how we integrated it into a real-time hand tracking system. In this section we present an overview of the tracking system. The input to the system is a depth image and associated grayscale image from the Intel RealSense 3D camera [22] (Figure 2a,b), and the output is a fully articulated hand model (Figure 2f), whose position and orientation in 3D space best fits the input data. While the depth image is used to match the hand model, the grayscale image (usually an IR image) is needed in order to detect and label fingers, as described in Section 4.2.2.

We use a kinematic model of the hand skeleton with twenty-six degrees of freedom, six for the root node and four for each finger. We employ a calibration module at runtime to adjust the lengths of each bone according to the proportions of the user's hand. The details of this calibration stage are beyond the scope of this paper. For simplicity, we assume that the bone lengths are constant. The kinematic constraints for plausible finger abduction/adduction and flexion/extension angles are set according to [19].

Our system solves an optimization problem that minimizes the distance between the input depth image and the hand model. We use a simple hand model inspired by a model from [12], composed of spheres and cylinders, that is attached (skinned) to the skeleton. Note, our approach can also be applied to other, more complex models, *e.g.* an accurate user-specific mesh model as in [20]. We solve the optimization problem by first finding a good initial guess for the kinematic parameters, and then improve it by applying the ICPIK algorithm.

The system is composed of two components; a blob segmentation component (Figure 3a) which identifies and tracks the blob of the hand; and a pose estimation component (Figure 3b), which computes the posture of the hand.

4.1. Blob segmentation

The process of extracting a blob corresponding to the user's hand begins with an over-segmentation of the depth image into super-pixels. The regions of the image are subdivided by thresholding the depth gradients. Subsequently, a set of heuristics are applied to merge and split the super-pixels into semantically meaningful blobs. These heuristics, for example, merge regions that overlap the previous frame's blob, if available, by a predefined amount; merge small adjacent regions that appear to be detached fingers; and filter out regions that are likely part of the user's arm. The system is initialized by classifying blobs as hand/not-hand using a Random Forest (RF) classifier with features as in Section 4.2.2. The blobs extracted in the current frame are then matched to those detected in the previous frame. The quality of the blob-to-blob matching is measured by mean depth comparison and by matching their contours.

4.2. Pose estimation

The first step of the pose estimation component is to estimate the rigid transformation of the hand between the pre-

vious and the current frames, as described in Section 4.2.1. Subsequently, a classifier is applied to identify regions of the blob that are likely candidates to be fingers, which are then used to generate plausible poses of the hand; see Section 4.2.2. Note, the candidate fingers are not dependent on the results of previous frame's tracking, and therefore provide an effective mechanism for error recovery. The *reinitialization module* (Section 4.2.3) generates a set of skeleton hypotheses, which are sets of kinematic parameters, that constitute initial guesses for the ICPIK algorithm.

4.2.1 Rigid motion estimation

The rigid motion of the hand from the previous to the current frame is estimated from a set of point pairs computed from the respective frames' depth blobs. We apply the RANSAC algorithm in order to find the best transformation based on triplets of the point pairs.

4.2.2 Finger detection and labeling

Candidate fingers are detected by applying an edge-detection algorithm to the grayscale image and searching for the locus bounded by two parallel edges approximately a finger-width apart, as shown in Figure 2c. We locate the "base" and the "tip" of each finger and estimate the radius R of the palm in the image. For each finger, we crop a $2R \times 2R$ square patch, centered at its base, and oriented toward the "base-to-tip" direction. HOG-like features are then extracted from these patches on the grayscale and depth images and used by an RF classifier to classify each finger as "Thumb", "Index", etc. The result is a set of probabilities for each finger, as shown in Figure 2d.

4.2.3 Hypothesis generation and error recovery

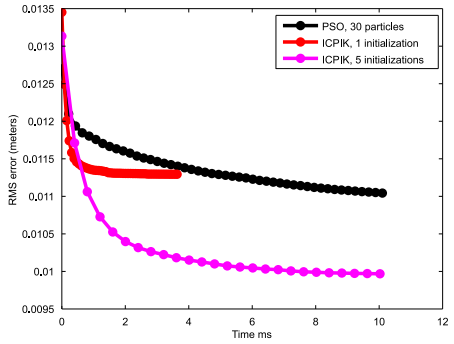
The reinitialization module generates a set of skeleton hypotheses that are subsequently passed as initial guesses to the ICPIK algorithm. When a hand is detected for the first time, a set of possible poses for the skeleton are generated from detected and labeled fingers (Section 4.2.2). When a blob was detected as a hand from the previous frame, the rigid transformation (Section 4.2.1) generates an additional hypothesis for the pose of the hand. The reinitialization module adds several more skeleton hypotheses, where each finger is assigned to one of the labeled fingers, assumed to be folded, or remains in its post-rigid pose. Each skeleton is given a score, indicating how well it fits the input, and the top K configurations are passed on to the module that performs ICPIK. The details of the score are described in the next section.

4.2.4 ICPIK refinement

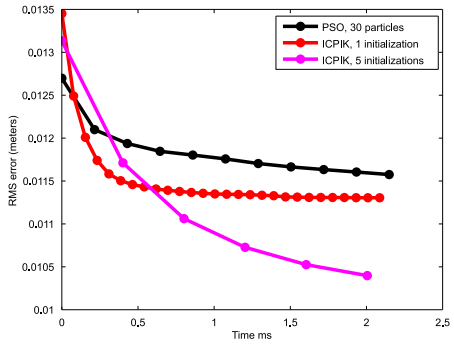
In the final step, we apply ICPIK algorithm to each skeleton hypothesis generated by the previous module. In order to apply ICPIK, we must first define correspondences between the virtual end-effectors of the articulated hand model and the virtual targets of the depth blob pixels. Recall that this step is analogous to matching the closest points between the source and target point clouds in an ICP implementation. However, we leverage the additional knowledge extant in the skeleton hypotheses regarding which regions of the depth blob correspond to fingers. We begin by projecting the skeleton's fingers onto the depth blob, to assign an appropriate finger label to each pixel. The labels are then propagated by the watershed transform to fill the entire blob. Next, we randomly subsample the pixels so that each label has at most L pixels assigned to it. A typical value is $L = 200$. We assign weights to the selected pixels so that the sum of the weights for each label equals one. Figure 2e shows an example of our labeling. These pixels are the virtual targets, and to each virtual target we associate a virtual end effector by choosing the closest point on the hand model with surface normal facing the camera. The root mean square (RMS) error between the virtual end-effectors and virtual targets is the error metric that is used to select the top K hypotheses in Section 4.2.3.

5. Results and conclusions

We evaluate the runtime performance and accuracy of the ICPIK algorithm on 3000 consecutive frames from six sequences of different people performing a variety of gestures in front of the camera. We compare a single-hypothesis version of the ICPIK algorithm to a PSO optimization similar to the one described in [12], and to a multiple-hypotheses ICPIK. In each case, the ICPIK component of Section 4 was replaced by the alternative optimization scheme, the PSO or the multiple-hypotheses ICPIK. However, the remainder of the system remained unchanged, so that at each frame each optimization scheme received the same initial state, as computed by the earlier components. We use PSO to optimize our objective function, rather than one described in [12]. The accuracy of each scheme was measured using the ICP objective error metric, *i.e.* root mean square (RMS) distance between sets of corresponding end-effectors and targets. Figure 4a is a plot of the ICP error as a function of the computation time in milliseconds for each algorithm. We use the hypothesis with highest score from the pose-detection module (Section 4.2.3) for initialization of the single-hypothesis ICPIK, and top five hypotheses from the pose-detection module for initialization of the PSO and ICPIK with multiple hypotheses. For the PSO scheme, we use 30 particles (six for each model), which are generated by randomly perturbing the



(a)



(b)

Figure 4. RMS Error vs. time of the ICPIK and the PSO optimizers. (b) shows the first few iterations in (a).

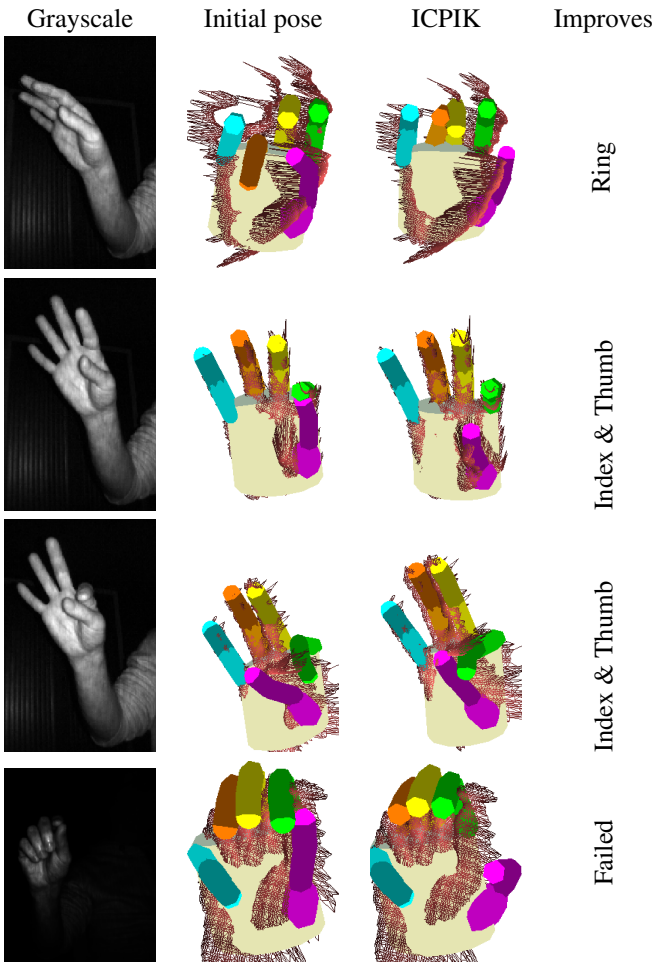


Figure 5. Qualitative results of the ICPIK algorithm. The left column shows an grayscale image, the middle column shows the initial configuration and the right column shows the result of the ICPIK algorithm. On the middle and left columns, the model is rendered along with a mesh of the input point-set in red.

same five states used to initialize the ICPIK. All experiments were performed on a laptop running an Intel Core-I7-4700hq CPU.

One can see in Figure 4b that the ICPIK algorithm typically converges within 8 iterations, requiring about $0.8ms$, while the PSO algorithm with 30 particles requires about $4ms$ to achieve the same error. The implementation of ICPIK based on five initial hypotheses further minimizes the error, which indicates that the single hypothesis ICPIK algorithm tends to converge to local minima and can benefit from multiple initializations. In the hand tracking system described in Section 4 we limit the number of iterations to 20, which requires $2ms$ of computation time for a single hypothesis ICPIK. For all twenty iterations, about 55% of the time ($1.1ms$) is spent computing the correspondences between the hand model and the depth image, while the rest of the time ($0.9ms$) is spent on computing the Jacobian matrix and the rest of the DLS solution.

Figure 5 displays qualitative results of the ICPIK algorithm, in which the output skeletons are rotated to provide better views. The first three rows show sample frames in which the algorithm successfully improved the initial hypothesis, while the fourth row shows a failure case. In general, given an adequate initial pose, the tracking system performs well, and the ICPIK scheme prevents the tracking from drifting. Occasionally, due to fast motions, noisy or imperfect depth image, or another source of error, the ICPIK can push the initial pose even further away from the desired solution, as in the last row of Figure 5.

Failures of the algorithm can be attributed to erroneous correspondences between the virtual end-effectors and their targets, and to inaccuracies in the hand model. Our method to find correspondences, as described in Section 4.2.4, relies on assigning correct labels to finger regions. Mislabeled fingers therefore generate incorrect correspondences, and distort the result. A promising direction for future research is, therefore, to improve the quality of the correspondences, for example, by ignoring the labels after a few iterations of the ICPIK, rejecting outliers, etc. Our hand model, shown in Figure 2f, is composed of spheres and cylinders. While the simplicity of the model lends itself to fast computation, it

does not come close to capturing the full complexity of the human hand. In particular, we have found that accurately modeling the articulation of the thumb is challenging, and, despite our best efforts, the thumb is responsible for many of the errors.

An additional improvement is to incorporate repulsive targets (as described in Section 3.1) to avoid various types of implausible configurations. Examples are self intersections, regions of the model that map to free-space of the depth map, and hand poses for which fingers occlude the palm, although the palm is visible in the camera's image. Properly defining and implementing these types of repulsive targets is another topic for future work.

In conclusion, we have presented an efficient articulated ICP algorithm, which incorporates the ICP problem into an inverse kinematics framework. The accuracy of the ICPIK algorithm is similar to the state of the art, while consuming significantly less computational resources. This is a result of the analytical evaluation of the Jacobian matrix coupled with an efficiently formulated optimization. Computational efficiency allows the algorithm to be run several times on different initial states, to improve accuracy. We have also demonstrated how the ICPIK algorithm achieves state-of-the-art results in a real-time hand tracking solution, and the approach can be similarly applied in the context of other articulated body problems, such as full-body skeleton tracking, and registration of 3D models to point clouds.

Acknowledgments. We gratefully acknowledge the contributions of Maoz Madmony, Dr. Shlomo Polonsky, Itamar Glazer, Moti Daniel, Chen Paz, Kfir Viente, Amit Bleiweiss, in implementing the real-time hand pose tracking system, as well as the help and support of all the former members of the Omek Interactive team.

References

[1] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, Feb. 1992. 1

[2] M. Bray, E. Koller-Meier, P. Mueller, L. Van Gool, and N. N. Schraudolph. 3d hand tracking by rapid stochastic gradient descent using a skinning model. In *CVMP*, 2004. 1, 2

[3] S. R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. Technical report, 2004. 3

[4] S. R. Buss and J.-S. Kim. Selectively damped least squares for inverse kinematics. *Journal of Graphics Tools*, 10:37–49, 2004. 2

[5] E. de Aguiar, C. Stoll, C. Theobalt, N. Ahmed, H.-P. Seidel, and S. Thrun. Performance capture from sparse multi-view video. In *SIGGRAPH*, 2008. 1

[6] G. Dewaele, F. Devernay, R. Horaud, and F. Forbes. The alignment between 3-d data and articulated shapes with bending surfaces. In *ECCV*, 2006. 2

[7] A. Erol, G. Bebis, M. Nicolescu, R. D. Boyle, and X. Twombly. Vision-based hand pose estimation: A review. *Computer Vision and Image Understanding*, 2007. 1

[8] J. Gall, C. Stoll, E. de Aguiar, C. Theobalt, B. Rosenhahn, and H.-P. Seidel. Motion capture using joint skeleton tracking and surface estimation. In *CVPR*, 2009. 1

[9] D. Grest, J. Woetzel, and R. Koch. Nonlinear body pose estimation from depth images. In *PR*, 2005. 2

[10] C. Keskin, F. Kirac, Y. E. Kara, and L. Akarun. Hand pose estimation and hand shape classification using multi-layered randomized decision forests. In *ECCV*, 2012. 2

[11] H. Li, R. W. Sumner, and M. Pauly. Global correspondence optimization for non-rigid registration of depth scans. In *Symposium on Geometry Processing (SGP)*, 2008. 1

[12] I. Oikonomidis, N. Kyriazis, and A. Argyros. Efficient model-based 3d tracking of hand articulations using kinect. In *BMVC*, 2011. 1, 2, 5, 6

[13] D. E. Orin and W. W. Scharader. Efficient computation of the jacobian for robot manipulators. *International Journal of Robotics Research*, 3(4):66–75, 1984. 2, 3

[14] S. Pellegrini, K. Schindler, , and D. Nardi. A generalization of the icp algorithm for articulated bodies. In *BMVC*, 2008. 2

[15] C. Qian, X. Sun, Y. Wei, X. Tang, and J. Sun. Realtime and robust hand tracking from depth. In *CVPR*, 2014. 1, 2

[16] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *International Conference on 3-D Digital Imaging and Modeling*, 2001. 1

[17] T. Sharp, C. Keskin, D. Robertson, J. Taylor, J. Shotton, D. Kim, C. Rhemann, I. Leichter, A. Vinnikov, Y. Wei, D. Freedman, P. Kohli, E. Krupka, A. Fitzgibbon, and S. Izadi. Accurate, robust, and flexible real-time hand tracking. In *CHI*, 2015. 1, 2

[18] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from a single depth image. In *CVPR*, 2011. 2

[19] E. Simo-Serra. Kinematic Model of the Hand using Computer Vision. Degree thesis, BarcelonaTech (UPC), 2011. 5

[20] J. Taylor, R. Stebbing, V. Ramakrishna, C. Keskin, J. Shotton, S. Izadi, A. Hertzmann, and A. Fitzgibbon. User-specific hand modeling from monocular depth sequences. In *CVPR*, 2014. 5

[21] J. Tompson, M. Stein, Y. Lecun, and K. Perlin. Real-time continuous pose recovery of human hands using convolutional networks. In *SIGGRAPH*, 2014. 2

[22] http://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html. 1, 5

[23] http://www.microsoft.com/en-us/kinectforwindows/. 1

[24] R. Y. Wang and J. Popović. Real-time hand-tracking with a color glove. In *SIGGRAPH*, 2009. 2

[25] W. Zhao, J. Chai, and Y.-Q. Xu. Combining marker-based mocap and rgb-d camera for acquiring high-fidelity hand motion data. In *SIGGRAPH/SCA*, 2012. 2