Controlling a computer by hand gestures

By bohuai jiang
< u4652552@ anu.edu.au>
A report submitted for the degree of Master of Computing of
Australian National University

Supervisor: Yuchao Dai, Laurent Kneip, Hongdong Li
COMP8740 Artificial Intelligence Project
Acknowledgments

I would like to thank my supervisors Yuchao Dai, Laurent Kneip, Hongdong Li and for providing his constant guidance, motivation and time for discussions and talks. Their expertise, patience, enthusiasm and friendship were highly appreciated and encouraged me greatly.
Abstract

This report presents a gradient base Iterative Closest Point Optimization (ICP) for hand pose tracking and estimation. We adopt a commonly used 26 degree freedom with 48 sphere geometric model for hand optimization. This project we started from scratch, since there is no existing pervious works or available libraries. We undertook a heavy workload on code developing. Due to the time constraints, we skip the steps of hand recognition (There are already exist several mature technique for recognition[7][14][5]), and focus on model construction, cost function computation and optimization which are the central of hand pose tacking and estimation. The result of using gradient base Iterative Closet Point Optimization indicate this approach cannot always find optimal value. But with implementation of stochastic optimization future improve its accuracy which we leave it in future works.

Key words: gesture control, ICP, gradient descent

List of Abbreviations

ICP    Iterative closest point
DOF    degrees of freedom
# Table of Contents

Acknowledgements ............................................................................................................. 2  
Abstract ................................................................................................................................. 2  
List of Abbreviations ............................................................................................................. 2  
1 Introduction .......................................................................................................................... 5  
   1.1. Overview ....................................................................................................................... 5  
   1.2. Objectives .................................................................................................................... 5  
   1.3. Related Works ............................................................................................................. 6  
2 Background ......................................................................................................................... 6  
   2.1 RGB-Depth camera ....................................................................................................... 6  
   2.2 Iteration Closest Point (ICP) ....................................................................................... 7  
3 Design and Implementation ................................................................................................. 8  
   3.1 Programing language .................................................................................................... 9  
   3.2 Convert depth image to 3d coordinate ......................................................................... 9  
   3.3 Build Hand model ......................................................................................................... 9  
   3.4 Compute cost function ................................................................................................. 13  
   3.5 Optimization .............................................................................................................. 14  
      3.5.1 Compute gradient ............................................................................................... 15  
      3.5.2 Gradient descent ............................................................................................... 17  
4 Experiment and Evaluation ............................................................................................... 20  
   4.1 Experiment Steps ......................................................................................................... 20  
   4.2 Testing Environment ................................................................................................. 21  
   4.3 Experiment Results ..................................................................................................... 24  
   4.4 Evaluation ................................................................................................................... 24  
5 Conclusion and Future Work ............................................................................................ 25  
   5.1 Conclusion ................................................................................................................... 25  
   5.2 Future Work ............................................................................................................... 26  
Appdex ................................................................................................................................ 27  
Reference .............................................................................................................................. 27
1. Introduction

1.1 Overview

With the advent of user experience success in smart phones and ultra-book, multi-touch not only shortened the distance between human and computer, but also enlighten people to a renewed interest in natural interfaces particularly in hand gestures. Hand gestures controls has great potential use in multimedia home platform and consumer electronics control system, it is believed as a new innovative user interface which resolve the complications of using remote control for domestic appliances[1].

Vision based hand gesture recognition and tracking has been intensely studied for 20 years [2,3,4,5,6,7]. However a robust method to recognize hand poses and gestures are still challengeable due to its complex and dexterous nature of hand articulations. Recent advances in commodity-level RGB-D cameras such as Kinect have greatly simplified those problems.

Beside some significant progress in recent year, the state-of–the–art approaches are often limited in some certain constraint, such as the work used discriminative salient points [27] received high accuracy on hand modeling, but consume high time complexity during optimization. Another approach using simple polygon model tracking, achieved real time performance [28][29], but require high usage on GUP. Moreover R.Y.Wang and J.Popovv [9] using color pattern base tracking obtain a high accuracy and real-time optimization, but restrict on wearing color gloves.

As the result to develop a simple model and robust optimization algorithm with low time complexity is the center of project.

1.2 Objectives

The Objectives of this project is aim to implement a real time human control interface based on hand gestures. But firs require a robust and real time human hand pose estimation and tracking system, with tracks a fully articulated hand under large view-point. The project base on RGB-
Depth camera. Due to time constraints we only concentrate on Model construction, cost function computation and optimization.

### 1.3 Related works


Relevant technique are studied, including Iterative Closest Point (ICP), Distance transform, linear search for gradient descent as well as some model building techniques.

Moreover a new program langue is learned: C++ with OpenGL lab including Vertex Buffer Object (VBO) method, which helped us to visualize our hand model and point cloud in this project.

## 2. Background

### 2.1 RGB-Depth camera

![RGB-Depth camera](image)

**Figure 1.** (a) RGB-Depth camera[32], (b) a depth image[31], (c) depth image in our data
RGB-Depth camera is a line of motion sensing input devices. It can produce dense 3D point cloud “picture” form an object, by shooting dense of infrared rays to the object and calculate the distance base on those reflection.

In compute vision, a lot of method requires on accurate and fast 3D scene analysis [22][29], such as geometry reconstruction, collision prevention, mixed reality, and gesture recognition. The acquisition of a range map by image base analysis is costly [9]. Nevertheless RGB-Depth with its fast photo speed, greatly simplified those problems.

2.3 Iteration Closest Point (ICP)

![Figure 2](https://via.placeholder.com/150)

*Figure 2 mapping M₁ to M₂, requires matrix transformation rotation.*

Iteration Closest Point is a common used algorithm in computer vision, to minimize the difference between two clouds of points (Finger 2). If given two point clouds $M_1$ and $M_2$. The basic ICP algorithm for mapping between those two point clouds is:

$$E = M_2R + T - M_1$$

Where $R$ denote matrix rotation and $T$ denote translation. The goal of ICP is to minimize $E$ by iteration. It can also be minimize by mathematical deduction.

In recent pose tracking and optimization approach, people often extends base ICP. In v.Ganapathi[11] and Qia[14] used point-model alignments ICP.

3. Design and Implementation
Due to the time constraints and relatively mature technique on object detection [17][18][19][20][21], we skip the process on hand detection and noise reduction, we directly used filtered data from Qian1[10] which are 320x240 depth images. Therefore there are only four steps in this project: 1. Preprocessing data, 2. build hand model. 3. Construct cost function. 4. Apply optimization

### 3.1 Programing Language

C++ is the only language used in this project and OpenGL is the only lab we used for visualization purpose. We select C++ since there are a lot Computer Vision library available on C++. However during the developing we discover there is no library available to this project.

### 3.2 Convert depth image to 3d coordinate

In this project due to the limit time contrasts, we directly used the depth data form Qain1[10]. This data is collected by Intel’s Creative Interactive Gesture Camera with the depth resolution 320x240 and intrinsic focal length 241.42 in millimeters. The data also have already filtered and denoised to hand shape silhouette. Those data is further converted into 3d point cloud as following:

\[
\frac{f}{Z} = \frac{u}{X} = \frac{v}{Y}
\]

where u, v are the coordinate from depth images. We denote the point cloud as \( P \)

### 3.3 Build Hand model

We adopt the commonly used 26 degrees of freedom (DOF) hand model [8][9], where 6 DOFs for global hand pose (translation $v_x$, $v_y$, $v_z$ and rotation $\alpha$, $\beta$, $\gamma$ at x, y, z axis), 4DOFs for each finger. (Finger 3,4).

For the global hand pose, 6 DOF, transformation $T'$ is calculate as $T' = PT_{xyz}R_{xyz}$ where $P$ is the coordinate i.e. $p = [x\ y\ z]^T$ and

$$T_{xyz} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{xyz} = \begin{bmatrix} \cos \beta \cos \gamma & \cos \gamma \sin \alpha \sin \beta - \cos \alpha \sin \gamma & \cos \alpha \cos \gamma \sin \beta + \sin \alpha \sin \gamma & 0 \\ \cos \beta \sin \gamma & \cos \alpha \cos \gamma + \sin \alpha \sin \beta \sin \gamma & -\cos \alpha \sin \gamma + \cos \alpha \sin \beta \sin \gamma & 0 \\ -\sin \beta & \cos \beta \sin \alpha & \cos \beta \sin \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
For each of fingers, there are 2 DOF in finger root (rotation $\theta_1$ along with palm and $\theta_2$ perpendicular to the palm) and 1 DOF in each finger joint (rotation $\theta$ perpendicular to the palm). To calculate the transformation of each fingers requires first compute the rotation axis. In this project the axis perpendicular to palm is calculated by first picking three points from DOF26 model, where two 2DOF $P_1, P_2$ from finger root (picked $P_1, P_2$ from current state of finger root and the one next to it) and one 1DOF $P_3$ from current state of finger next to it finger root. Then compute direction vectors:

$$V_1 = P_2 - P_1, \ V_2 = P_3 - P_1.$$ Eventually the axis is compute as:

$$[u_z \ v_z \ w_z] = V_1 \times V_2.$$ For compute the axis along with palm, we first calculate angle $\alpha$ between $V_1, V_2$:

$$\alpha = \cos^{-1}((V_1 \cdot V_2)/(|V_1||V_2|)).$$

Then rotate $V_1 = [u \ v \ w]$ at $[u_z \ v_z \ w_z]$ for $\alpha$ angle:

$$[u_x \ v_x \ w_x]$$

$$= \begin{bmatrix}
    u_x(u_z \ u + v_z \ v)(1 - \cos \alpha) + (u_x^2 + v_x^2 + w_x^2)ucosa + \sqrt{u_x^2 + v_x^2 + w_x^2}(-w_z v + v_z w)\sin \alpha \\
    v_x(u_z \ u + v_z \ v)(1 - \cos \alpha) + (u_x^2 + v_x^2 + w_x^2)vcosa + \sqrt{u_x^2 + v_x^2 + w_x^2}(w_z u - u_z w)\sin \alpha \\
    w_x(u_z \ u + v_z \ v)(1 - \cos \alpha) + (u_x^2 + v_x^2 + w_x^2)wcosa + \sqrt{u_x^2 + v_x^2 + w_x^2}(-v_z w + u_z v)\sin \alpha \\
    \end{bmatrix}$$

Base on those rotation axis $[u_z \ v_z \ w_z]$ we calculate the rotation transformation $T$ by $\theta_1$ angle as:

$$T = \begin{bmatrix}
    u_z(u_z \ x + v_z \ y)(1 - \cos \theta_1) + (u_x^2 + v_x^2 + w_x^2)x\cos \theta_1 + \sqrt{u_x^2 + v_x^2 + w_x^2}(-w_z \ y + v_z \ z)\sin \theta_1 \\
    v_z(u_z \ x + v_z \ y)(1 - \cos \theta_1) + (u_x^2 + v_x^2 + w_x^2)y\cos \theta_1 + \sqrt{u_x^2 + v_x^2 + w_x^2}(w_z \ x - u_z \ z)\sin \theta_1 \\
    w_z(u_z \ x + v_z \ y)(1 - \cos \theta_1) + (u_x^2 + v_x^2 + w_x^2)z\cos \theta_1 + \sqrt{u_x^2 + v_x^2 + w_x^2}(-v_z \ x + u_z \ y)\sin \theta_1 \\
    \end{bmatrix}$$
Base on those rotation axis \([u_x \ v_x \ w_x]\) we calculate the rotation transformation \(T\) by \(\theta_2\) angle as:

\[
T = \begin{bmatrix}
    u_x(u_x x + v_x y)(1 - \cos\theta_2) + (u_x^2 + v_x^2 + w_x^2)\cos\theta_2 + \sqrt{u_x^2 + v_x^2 + w_x^2}(-w_x y + v_x z)\sin\theta_2 \\
    u_x^2 + v_x^2 + w_x^2 \\
    v_x(u_x x + v_x y)(1 - \cos\theta_2) + (u_x^2 + v_x^2 + w_x^2)\cos\theta_2 + \sqrt{u_x^2 + v_x^2 + w_x^2}(w_x x - u_x z)\sin\theta_2 \\
    u_x^2 + v_x^2 + w_x^2 \\
    w_x(u_x x + v_x y)(1 - \cos\theta_2) + (u_x^2 + v_x^2 + w_x^2)\cos\theta_2 + \sqrt{u_x^2 + v_x^2 + w_x^2}(-v_x x + v_x y)\sin\theta_2 \\
    u_x^2 + v_x^2 + w_x^2
\end{bmatrix}
\]

Similarly rotation transformation \(T\) on 1 DOF for \(\theta\) angle is calculated as:

\[
T = \begin{bmatrix}
    u_x(u_x x + v_x y)(1 - \cos\theta) + (u_x^2 + v_x^2 + w_x^2)\cos\theta + \sqrt{u_x^2 + v_x^2 + w_x^2}(-w_x y + v_x z)\sin\theta \\
    u_x^2 + v_x^2 + w_x^2 \\
    v_x(u_x x + v_x y)(1 - \cos\theta_2) + (u_x^2 + v_x^2 + w_x^2)\cos\theta + \sqrt{u_x^2 + v_x^2 + w_x^2}(w_x x - u_x z)\sin\theta \\
    u_x^2 + v_x^2 + w_x^2 \\
    w_x(u_x x + v_x y)(1 - \cos\theta_2) + (u_x^2 + v_x^2 + w_x^2)\cos\theta + \sqrt{u_x^2 + v_x^2 + w_x^2}(-v_x x + v_x y)\sin\theta \\
    u_x^2 + v_x^2 + w_x^2
\end{bmatrix}
\]

Since there are totally 21 joint point in DOF26 model. Its time complexity of computing transformation on 6DOF is \((21)\). And for each finger the time complexity of compute rotation on \(\theta_1 \ \theta_2\) and \(\theta\) is \(O(5*(1+2+3)) = O(30)\). Therefore the total time complexity of transformation is \(o(51)\) which is instantaneous, thus it is feasible to apply.

The DOF26 model then been transformed in to a simple geometric model for fast computation of distance and intersection purpose. Simple geometric models have already been used into many gesture tracking approach such as in human body tracking, human body often modeled as mixture of sphere and cylinders [11]. In this work we adopt form [10] use the simplest sphere set (total number of 48 sphere) representation as illustrated in Figure 1. The number of spheres for each part is manually specified: 6 for each finger and 18 for palm. The sphere radius are fix values and adopted from CVPR14 data, but the center of each sphere is depends on
transformation of DOF26 model. Thus our hand model may limit on general data. Nevertheless using user-specific hand modeling [12] should conquer the problem and further improve its performance.

We denoted the geometric model denoted as $M(\theta) = \{s_i\}_{i=1}^{48}$, where each sphere $s$ has two parameters: center $c(\theta)$ and radius $r$, denote as $s = \{c(\theta), r\}$.

### 3.4 Compute cost function

For real time tracking purposes, we adopt a simple cost function form Qian[10], as following:

$$E = \lambda \sum_{p \in \text{sub}(P)} D(p, s_{x(p)})^2 + \sum_i B(c_i, D)^2 + \sum_{i,j} L(s_i, s_j)^2$$

(1)

The cost function is composed in three parts for three purposes: 1. Measure the discrepancy between sphere $M$ (form geometric model) and closet data point $P$ (from cloud point). 2. Provide punishment mechanism on geometric model for which lie outside of point cloud. 3. Provide punishment mechanism for inter-collision between spheres in geometric model.

The first term $D(p, s_{x(p)})^2$ aim to measure the discrepancy between $M$ and $P$. In order to reduce the computational complexity, 256 samples is randomly selected out of $|P|$, and denote as $\text{sub}(P)$. For each point, $x(p)$ denote the closest sphere to this point. And $D(\cdot)$ is the distance from that point to the sphere surface. We assume random selected sample are often distributed, but we cannot avoid exceptional conditions occurring. Therefore using other sample techniques such as systematic sampling [14] or stratified sampling [15] should future improve cost function reliability. Euclidian distance is used to calculate the difference,

$$D(p, s) = abs(||p - c|| - r)$$

(2)

The second term $B(c_i, D)^2$ aims to ensure all sphere of hand model are lie inside the point cloud (Figure 3). Each sphere center is project on to depth math $D$, denote as $j(c)$. The penalty is
given if the depth at \( j(c) \) is behind \( j(c) \) or if there is not depth at \( j(c) \). If there is no depth at \( j(c) \), that is \( j(c) \) is outside of depth image, it received penalty \( d \) which is the shortest between \( j(c) \) to silhouette. We used distance transforms with Euclidean Distance Matrix to efficiently compute the distance to silhouette [16]. since the distance is compute in pixel we future converted it to millimeters using the average input depth. Its formula is defined as

\[
B(c, D) = \begin{cases} 
\max(0, D(j(c)) - c_z) & \text{if depth map has depth at } j(c) \\
\text{dist}(j(c), \text{silhouette of } D) & \text{other wise}
\end{cases}
\]  

(3)

The last term \( L(s_i, s_j)^2 \) focus to void self-collisions. Since most of collision occurs during optimization are between fingers, we only test finger collision for efficiency. It formula is defined as:

\[
L(s_i, s_j) = \max(r_i + r_j - \|c_i - c_j\|, 0) 
\]

(4)

We also multiply \( \lambda = |M|/|sub(P)| \) to first term, so that the first term magnitude is same as other terms.

This cost function is simple and efficient. Its first term time complexity is \( O(|M|_{sub(P)}) \), and its rest of terms complexity only depends on \( |M| \), therefor the totally time complexity of cost function is \( O(|M|_{sub(P)}) \).

3.5 Optimization

Tracking object base on local optimization from initial hand pose of last frame are famous in the state-of-the-art approaches [17][13][12]. For our point-model alignment tasks, Iterated Closeted Point is widely used method [21][22]. We adopt this approach and apply gradient descent base ICP for our optimization, which is converges fast and easy to be understand.
3.5.1 Compute gradient

3.5.1.1 Base on Numerical Jacobian

Numerical Jacobian is used for computing gradient of 26 parameters in DOF26 with $E$, which is limit base gradient calculation and easy to imply for complex cost function. Its formula as follows:

$$\nabla F = \begin{vmatrix} \nabla f_1 \\ \nabla f_2 \\ \vdots \\ \nabla f_{26} \end{vmatrix} = \begin{vmatrix} \frac{E - \delta E_1}{\delta_1} \\ \frac{E - \delta E_2}{\delta_2} \\ \vdots \\ \frac{E - \delta E_{26}}{\delta_{26}} \end{vmatrix}$$

In this formulate each parameter gradient are represent as $\nabla f_i$, $E$ is the original energy (cost) before transformation, and $\delta E$ is the energy (cost) after changing the parameter with $\delta_i$ value:

$$\nabla F_i = computeCost\left(\text{transform}\left(\begin{vmatrix} p_1 \\ \vdots \\ p_i + \delta_i \\ \vdots \\ p_{26} \end{vmatrix}, \text{DOF26}\right)\right)$$

To ensure the accuracy the value of $\delta$ need to be small. In this project $\delta_i$ are set manually bas on the experiment result. Its pseudo-code as following

---

**Pseudo-code** for Computing Jacobian Matrix

```plaintext
ComputJacobianMatrix(deltaMatrix, transformMatrix, DOF26)

1  tempTransformMatrix ← transformMatrix
2  i ← 0
3  initialize(deltaEMatrix)
4  for each $p \in$ transformMatrix do:
5      np ← $p +$ deltaMatrix[i]
6      newTransformMatrix ← update p to np in tempTransformMatrix
7      newDOF26 ← transform(newTransformMatrix, DOF26)
8      deltaE ← computeCost(newDOF26)
```
The time complexity of computing Jacobian matrix is $O(|\text{transformMatrix}| \times |\text{sub}(p)| \times |M|)$, where $|\text{sub}(p)| \times |M|$ is the time complexity of computing cost function.

Since our cost function is intensely rely on the pair up relations (relations between point cloud and Model), which are calculated by closest distance. Our cost function produces discrete value during Model transformation. As the result the value of $\delta$ need to be extremely careful chosen in order to get accurate gradients, nevertheless numerical Jacobian approach turns provides quite high accuracy gradients during our experiment.

**3.5.1.2 Base on Derivative formula**

Since Numerical Jacobian approach are more or less provide inaccurate gradients. Also it is difficult to find a general $\delta$ works on all depth image data. Therefore we provided another approach on calculating gradient i.e. derivative approach. This section briefly describes the derivative approach for computing gradient. Due to the time constrains this approach has not yet been implemented in our experiment, and we leave the implementation as future work.

Based on our cost function, the transformation function for checking distance (terms 1) of a single sphere in $M$ is:

$$E_1 = abs\left(\left\| p - f(c, \begin{vmatrix} p_1 \\ \vdots \\ p_{26} \end{vmatrix}) \right\| - r \right)$$

Where the $f(\cdot)$ is the transformation function base on DOF26. For a matrix $M$ to transform into a matrix $M'$ the transformation function gives:
Thus product rule can be applied on computer transformation to get its derivate. The detail derivate of $M'$ on $R_{xyz} T_{xyz} R_{12} R_3 R_4$ using product are showed in Appendix, here we use $df(\cdot)$ to represent the derivate of $f(\cdot)$.

The product rule can also apply to Euclid distance. Given Euclid distance function:

$$D = (x - f(c)_x)^2 + (y - f(c)_y)^2 + (z - f(c)_z)^2$$

It gradient of $f(c)$ is:

$$dD/d(f(c)) = 2(x - f(c)_x)df(x)_x + 2(y - f(c)_y)df(c)_y + 2(z - f(c)_z)df(c)_z$$

Therefor the gradient of first terms in cost function gives:

$$E' = \left\| p - f(c), \begin{bmatrix} p_1 \\ \vdots \\ p_26 \end{bmatrix} \right\| \ast dD/d(f(c))$$

Product rule can also been used for calculating the gradient of lie inside checking (term 2) and collision checking (term 3) in cost function. Their methods are similar with term1. The final gradient of $E$, $\nabla E$ is:

$$\nabla E = \lambda \sum_{p \in \text{sub}(P)} D(p, f(s_{x(p)})) \frac{dD}{d(f(s_{x(p)}))} + \sum_{i} B(f(c_i), D) \frac{dB}{d(f(c_i))} + \sum_{i,j} L(f(s_i), f(s_j)) \frac{dL}{d(f(s_i), s_j)}$$

Although this Derivative Approach is hard to be deduce, but it can produces high accuracy gradient and it does not require inputting any manual parameter. Since the derivative function is deduce from cot function. Thus the time complexity of computing gradient using this approach is equivalent to the time complex of cost function i.e. $O(|M|^*|\text{sub}(p)|)$, which is less than Numerical Jacobian approach.

### 3.5.2 Gradient descent
Gradient descent is a widely used multivariable function optimization technique. In recent year, it has been used a lot in object tracking optimization\cite{23}\cite{24}\cite{25} and has been proved feasible for tracking optimization. In this project we adopt general gradient descent to find the optimal values of DOF26. Its formula as:

\[
 p_i := p_i - \lambda \nabla E
\]

where \( \lambda \) is the searching rate and \( \nabla E \) is the derivatives of our cost function. To ensure \( E \) is reduce in each step of update \( p_i \), we let \( \lambda \) able to self-adjust during iteration. For each of update, if new \( p_i \) will increasing the cost, then halt update and reduce \( \lambda \) until the new \( p_i \) will decrease the cost. If there is no \( \lambda \) could future decrease the cost, then set \( \lambda \) as 0. The pseudo-code as following:

---

**Pseudo-code** for self-adjust lambda

```
SelfadjustLambda(\lambda , transformationMatrix, DOF26,i, \nabla E_i)
1    E ← computCost(DOF26)
2    tempTran ← transformationMatrix
3    count ← 0
4    while true do:
5        tempDOF26 ← DOF26
6        tempP ← transformationMatrix[i] - \lambda \nabla E_i
7        Update tempP to tempTran
8        Transform(tempTran,tempDOF26)
9        newE ← computCost(tempDOF26)
10       If newE < E then:
11           break
12       else
13           \lambda ← \lambda /2
14       If count > 100
15           return 0
16       count++
17       return \lambda
```
input: initial value of $\lambda$, transformation matrix, DOF26 model, gradient at index $i$ and index $i$.

Time complexity is $O(100^{|M|\text{sub}(p)})$.

We also increasing the $\lambda$ value during the gradient descent, in the case if converge seep is slow. The pseudo-code of gradient descent as following:

---

**Pseudo-code** for gradient descent

GradientDescent($\lambda Matrix$, $\delta Matrix$, DOF26, transformMatrix)

1. iteration ← 0
2. $\nabla E$ ← ComputJacobianMatrix($\delta Matrix$, transformMatrix, DOF26)
3. afterE ← 0
4. preE ← 0
5. while true do:
6.   tempDOF26 ← DOF26
7.   iteration ++
8.   if iteration > 5
9.     $\lambda Matrix$ ← 2*$\lambda Matrix$  
10.    increasing $\lambda Matrix$ if its converge speed is slow
11.   E ← computCost( (DOF26)
12.   for i in range [1..26] do:
13.     $\lambda Matrix[i]$ ← SelfadjustLambda($\lambda Matrix[i]$ , transformMatrix,DOF26,i, $\nabla E[i]$)
14.     transformMatrix[i] ← transformMatrix[i] - $\lambda Matrix[i]$ * $\nabla E[i]$
15.   Transform(transformMatrix,tempDOF26)
16.   afterE ← computeCpst(tDOF26)
17.   If abs(preE-afterE) <= 0.1
18.     break
19.   preE ← afterE
20. return transformMatrix

---

The final gradient descent ICP as following figure 5 demonstrate it performances in each iteration:

---
Pseudo-code ICP

\[
\text{ICP}(\lambda Matrix, \delta Matrix, \text{DOF26, transformMatrix})
\]

1. \( \text{pertrans} \leftarrow \text{null} \)
2. \( \text{while} \) true \( \text{do}: \)
3. \( \text{trans} \leftarrow \text{GradientDescent}(\lambda Matrix, \delta Matrix, \text{DOF26, transformMatrix}) \)
4. \( \text{if} \) different between \( \text{trans} \) and \( \text{pertrans} < \text{threshold} \)
5. \( \text{break} \)
6. \( \text{pertrans} \leftarrow \text{trans} \)
7. \( \text{return} \) \( \text{trans} \)

---

Figure 6 demonstrated the gradient descent based ICP on hand estimation. The top left model is the initial hand pose. We want to match the hand model with the point cloud behind it. The right bottom is the final output after 8 iterations.

### 4. Experiment and Evaluation

This experiment focused on testing robustness of gradient descent base ICP. The ground truth are selected form Qia[10]². Toward to this purpose we test ICP performance base on two factors: time consumption and accuracy (denote by E which is energy difference with ground truth).

---

² Available at http://research.microsoft.com/en-us/people/yichenw/
4.1 Experiment Steps

We divide this experiment into two part: First. Test the ICP optimization performance base on last frame initialization. Secon,. Test ICP optimization performance base on changing DOF26 variables for current frame pose initialization.

Our gradient descent base ICP is design for local optimization for an initial hand pose, which is from last frame. As the result the first experiment is conducted, its step as follows:

4. Initialize hand model base on the ground truth of frame \(a\).
5. Run ICP optimization for its next frame \(b\).
6. Compare ICP result with ground truth of \(b\).
7. Calculate error \(E\), which is difference between cost function of ground truth with cost function of ICP result.

On the other hand, we designed second experiment, aim to study the effect of different pose initialization on ICP optimization. The step as follows:

1. Add random noise between \(-1^\circ\) to \(1^\circ\) to all rotation variable from DOF26, and Add random value between -1 to 1 to all translation variable from DOF26
2. Run ICP can and compute error \(E\), where \(E = \text{ICPCost} - \text{originalCost}\)
3. Increasing random value range. Repeat step 1 and 2.

4.2 Testing Environment

Experiments are conducted via Microsoft Visual Studio. Table below presents details of hardware and software environment
<table>
<thead>
<tr>
<th>Hardware/Software</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Visual Studio</td>
<td>Version 12.0</td>
</tr>
<tr>
<td>Operating system</td>
<td>Windows 8</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel(R) Core(TM) i7-3632QM 2.20GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>8.00 GB</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>1TB</td>
</tr>
</tbody>
</table>

Table 1. Hardware and software information

### 7.2 Experiment Results

In our experiment, we set $\lambda$ of all rotation parameters as 0.01, and $\lambda$ of all translation parameters as 1. Similarly we set $\delta$ of all rotation parameters as 0.01, and set $\delta$ of all translation parameters as 0.1. Since it transformation is calculate in millimeters, thus we believe translation with 0.1 millimeters is small enough to represent its gradient.

**Initialize from last frame**

We have tested 20 frames form Qia[14] data and compared their ground truth with we our IPC result. The average time consumption for ICP to converge is about 0.437 second. The average E error rate is about 0.8147. Here we collected some 5 iconic results out of 20 for evaluation.

<table>
<thead>
<tr>
<th>Last frame</th>
<th>current frame</th>
<th>Initialization</th>
<th>After ICP</th>
<th>Time</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame 6</td>
<td>frame 7</td>
<td></td>
<td></td>
<td>0.3</td>
<td>0.817</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>second</td>
</tr>
<tr>
<td>frame 6</td>
<td>frame 7</td>
<td></td>
<td></td>
<td>0.2</td>
<td>0.905</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>second</td>
</tr>
</tbody>
</table>
frame 8   frame 9

<table>
<thead>
<tr>
<th>Frame</th>
<th>Change Variable on Current Frame</th>
<th>0.1 second</th>
<th>0.127 second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame 26  Frame 27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame 36  Frame 37</td>
<td>0.26 second</td>
<td>0.9134 second</td>
<td></td>
</tr>
<tr>
<td>Frame 16  Frame 17</td>
<td>0.5 second</td>
<td>24.989 second</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Performance of initialize hand pose form last frame

**Change variable on current frame**

The next experiment, we used frame 1 (Figure 4) in dataset as our initial hand pose. Then adding random noise to each of its DOF26 parameters. The range of random noise are manually specified. In graphs 1 and 2, “Range of random variables” $v$, represents random select a variable form $-v$ to $v$. E.g. random of random variables 8 means the random variable are select form range -8 to 8. The “time consumption” in this experiment is measured in seconds.
Graph 1. Performance of increasing random range from 1 to 20

Graph 2. Performance of increasing random range from 5 to 100

4.6 Evaluation

Initialize from last frame

The overall performance of experiment 1 is acceptable. Using gradient descent base ICP optimization base on last frame pose initialization, it can find optimal value in most of case and product with short time consumption.

However ICP also failed and tracked in some local optimal value, such as the result in frame 16 and 17. That’s mainly coasted by the limitation of gradient descent which cannot guarantee to find global optimal variable. In our implementation of gradient descent, we try to minimize thus problem by auto increasing size of $\lambda$ during it iteration (explained in section 3). To further improve its performance, stochastic optimization such as Partial Swarm Optimization (PSO) is necessary to be implied.

Change variable on current frame
Buy observing the result of Experiment 1, it is clear that adding small noise into initial hand pose, ICP optimization still can achieve optimal results. However, with increasing size of random noise, its time consumption for converge turns to increasing. Converge time can be improved by increase the size of $\lambda$, but it may produce low accuracy.

By adding great range of random noise (in Graph 2), ICP cannot produce accurate output. However with increasing noise size, ICP time consumption turn to decrease. That indicate ICP is tracked into a local optimal value.

During the experiment, we discover ICP often produced a counter-intuitive hand pose (figure) when it tracked into a local optimal. That cause by our DOF26 hand model, there is no constraint for our rotation angles. This inspire us to use possibility base approach to limit local optimal. Using possibility base approach like naive bays or Hidden Marko model to pre-train set of hand pose. Then during the gradient changing value of $\lambda$ will also depends on it possibility of the hand pose it will be formed. Thus $\lambda$ value become more intelligent.

5. Conclusion and Future Work

5.1 Conclusion

This project controlling computer by hand gestures is developed via C++ and openGL. We started it from scratch. Since there is no previous work or available library, we took a heavy work on code developing. Due to the time constrains we only accomplish the most center part of Hand tracking and estimation, which are hand model construction, cost function computation and gradient descent based ICP optimization.

From our experiment we discovered that optimization used gradient descent base ICP, is acceptable to the hand pose for last frame initiation. But performed poorly on random initialization. Thus gradient base of ICP approach are not feasible for hand tracking and pose estimation. Nevertheless its limitation should be improve using stochastic optimization.
Moreover, gradient descent base ICP may also be improved using possibility base approach to select $\lambda$. And we leave it to future work.

In conclusion, although gradient descent base IPC is not feasible for hand tracking and pose estimation, but it has a great potential of improvements. As the result we believe our project can be used as a starting point for future optimization study.

### 5.2 Future Work

Future work should first replace “random selection” in cost function to some other sample technique, such as systematic sampling [14] or stratified sampling [15]. Hand model need also to be improved, in order to represent general data, supervised learning technique such as Random decision forests [30] can be used in this approach.

Lastly in optimization, techniques like POS, PCI-POS or other stochastic optimization need to implement to improve optimization accuracy.

### Appendix

Derivate of $M'$ on $R_{xyz}$ $T_{xyz}$ $R_{12}R_{3}R_{4}$

$$d(M')/dR_{xyz} = \begin{cases} 
MR'_{xyz} & \text{if it at palm} \\
(MR_{xyz} + T_{xyz})MR'_{xyz} & \text{if it round a finger root} \\
(MR_{xyz} + T_{xyz})^2 R_{12}R_{3} & \text{if it round finger mid joint} \\
(MR_{xyz} + T_{xyz})^3 R_{12}^2 R_{3}^2 R_{4} & \text{if it round finger first joint}
\end{cases}$$

$$d(M')/dT_{xyz} = \begin{cases} 
1 & \text{if it at palm} \\
(MR_{xyz} + T_{xyz})R_{12} & \text{if it round a finger root} \\
((MR_{xyz} + T_{xyz})R_{12})R_{3} & \text{if it round finger mid joint} \\
((MR_{xyz} + T_{xyz})R_{12})R_{3}R_{4} & \text{if it round finger first joint}
\end{cases}$$
\[ d(M')/dR_{12} = \begin{cases} 
0 & \text{if it at plam} \\
(MR_{xyz} + T_{xyz}) & \text{if it round a finger root} \\
((MR_{xyz} + T_{xyz})^2 R_{12}) R_3 & \text{if it round finger mid joint} \\
((MR_{xyz} + T_{xyz})^2 R_{12}^2) R_3 R_4 & \text{if it round finger first joint}
\end{cases} \]

\[ d(M')/dR_3 = \begin{cases} 
0 & \text{if it at plam} \\
0 & \text{if it round a finger root} \\
(MR_{xyz} + T_{xyz})R_{12} & \text{if it round finger mid joint} \\
((MR_{xyz} + T_{xyz})^2 R_{12}^2) R_3 R_4 & \text{if it round finger first joint}
\end{cases} \]

\[ d(M')/dR_4 = \begin{cases} 
0 & \text{if it at plam} \\
0 & \text{if it round a finger root} \\
0 & \text{if it round finger mid joint} \\
((MR_{xyz} + T_{xyz})R_{12}) R_3 & \text{if it round finger first joint}
\end{cases} \]

Reference


If any question, please let me know.

Presentation:

- Review of the project
- Introduction to the research field
- Project objectives

The project involves the development of software and the implementation of a real-time hand tracking system. This system will be used in fields such as virtual reality and interactive systems. The project will focus on the tracking of hand gestures in real-time, which will be implemented using a combination of sensors and algorithms.

Project Description:

(1) Research on 3D hand pose estimation and tracking with a RGB-D camera
(2) Development of a software component for hand gestures
(3) Development of a user interface for controlling the computer

Learning Objectives:

- Understanding the principles of hand tracking
- Developing software for hand tracking
- Designing user interfaces for controlling the computer

Project Supervisor: Dr. Vichio C.J.  (Laurent Kapf)

See the following for the information needed for your enrollment. Hardware and software are welcomed to make comments.