Universitat Autònoma de Barcelona

Master in Computer Vision and Artificial Intelligence

Report of the Master Project

Option: Computer Vision

Spherical Blurred Shape Model for Hand Pose Recognition

Author: **Miguel Pousa**

Advisor: **Sergio Escalera,**

Jordi Gonzalez

Acknowledgements

This work has been done in an internship at Oblong. I want to thank them for the unique opportunity that they gave us. And also thanks the CVC for make it possible. I am indebted with my advisors Sergio Escalera and Jordi Gonzalez for their comprehension, the dedication they put into this project and the great talks that we have shared. I also want to thanks my colleague Oscar Lopes that have been always there to share advices and comments. I also don't forget about Miguel Sánchez who has been always supportive and helpful. To all of you thank you.

ABSTRACT

This work presents a Hand Pose Recognition system from 3D pointclouds based on an object recognition approach that can be used in Human Computer Interaction applications. The system works in real time in a normal laptop. For the description part of the system this work presents a novel 3D descriptor called Spherical Blurred Shape Model (SBSM) that encodes the shape of an objet through an spherical decomposition. Our evaluation shows high accuracy in a real test set, and investigates the effect of the different system parameters.

Keywords: *Kinect camera, Depth sensors, Human Computer Interaction, Hand Pose Recognition, 3D descriptors, real time system*

Contents

1 Introduction **1**
1.1 Objectives . . . 2

2 System Overview **3**

3 Detector **6**

4 SBSM descriptor **8**
4.1 Grid definition . . . 8
4.2 Descriptor Computation . . . 9
4.3 Rotation invariant descriptor . . . 10

5 Classification **13**

6 Experiments **15**
6.1 Influence of the number of SBSM grid divisions . . . 15
6.2 Hyperparameter selection with cross validation . . . 17
6.3 The effect of the neighbors propagation in SBSM . . . 17
6.4 Computational cost . . . 18
6.4.1 Descriptor . . . 18
6.4.2 Detector . . . 19
6.4.3 Classifier . . . 19
6.5 Oblong integration . . . 19

7 Conclusions **22**

A Oblong concepts **23**

Chapter 1

Introduction

The human hand is capable of performing multitude of different poses, being able to recognize thoses poses is very useful for several applications, such as Human Computer Interaction, gestural communication and robotics.

Traditionally the hand pose recognition problem is addressed using normal cameras that capture 2D rgb or graylevel images. These approaches usually rely on color information [1] , shape information [2], or both [3] to segment the hand and describe it. Using 2D images makes the pose estimation a very changelling problem because of the great variability of hand poses that can have a similar 2D image. In order to surpass this difficulty it is common to infer 3D information [4] from shading, movement or other cues, but this greatly increases the computational complexity, making it difficult to allow for a real time system.

In the present days with the advent of the kinect-like sensors, RGBD images can be obtained with a resolution of 640x480 pixels at 30 frames per second. These sensors can capture depth images in a range from 0.8 to 3.5 meters and are more affordable than other kind of technologies with the same features.

With depth data the use of 2D shape descriptors like the Histogram of Oriented Gradients or the Haar-like filters do not suit the particularities of the 3D information, since they do not encode the spatial relation that exist between points in the depth axis.

Several 3D descriptors have been presented in the literature. The Feature Point Histogram [5] is a local descriptor that is used to recognize points conforming a plane, a cilinder or other geometric primitives. This geometric information is useful when classifying every day objects like cans, glasses or doors and can be employed in scene analysis. The Fast Point Feature Histogram (FPFH) [6], optimizes the PFH computation to make it usable in real time 3D registration applications.

Other proposed feature for 3D pointclouds is the set of normals of the surface defined by a given point and its neighbors [7][8]. The normals can be useful to recognize 3D objects as they encode the implicit surface that neighboring points define, yet they depend on the density of the underlying points to give accurate results.

The spherical harmonics [9] can be used to make 3D descriptors invariant to rotation [10] or directly as features [11].

Conformal factors [12] measure the relative curvature of a vertex given the total curvature. The result can be viewed as a vector, that is not only invariant to rigid body transformations, but also to changes in pose (given that the changes in pose do not modified the edge lengths). Yet conformal factors require vertex computation which can be a problem in frames from kinect-like devices, because they frecuently present holes that difficult a correct vertex calculation.

A descriptor based on FPFH suitable not only to object recognition but to pose estimation is the Viewpoint Feature Histogram (VFH) [13], which combines an extended version of FPFH with statistics between the viewpoint and the surface normals on the object.

Although new descriptors suitable for describing 3D information are appearing, still the research on 3D point cloud descriptors is an open issue. This work proposes a novel descriptor called Spherical Blurred Shape Model (SBSM) that is based on [15]. The SBSM is a global descriptor that encodes the shape of an object with an spherical decomposition.

Our system employs the SBSM descriptor to encode the shape of the hand. We have integrated the descriptor with a statistical classifier in order to recognize the different hand poses and have obtained a high recognition rate in our test set.

This Master Thesis has been made in an internship at Oblong, a company that develops futuristic Human Computer Interaction systems. Oblong has made previous work in hand pose recognition. In [14] they developed an approach based on an extrema detection followed by a combination of global and local features and a random forest classifier.

1.1 Objectives

Develop and implement the SBSM pointcloud descriptor and study their properties. Design a Hand Pose Recognition System using the SBSM descriptor that works in real time and is suitable for Human Computer Applications. Study the integration of this system into the stack of technologies that Oblong have and implement an example.

The rest of this report is organized as follows: A complete review of the system that describe the links between the different parts is discussed in section 2. Each of the system parts are described in detailed in its own section, the detector 3, the descriptor 4 and the classificator 5. The results of this work are presented in section 6 and the conclusions in section 7. An appendix with explanations about the particularities of the Oblong platform is developed in A. We conclude with the bibliography.

Chapter 2

System Overview

We treat the hand pose recognition problem with an object recognition approach, that is, we decompose the problem into three subproblems: detection, description and classification. In the detection stage the hands are extracted from the scene. Then in the description stage the hands are described to extract meaningful information that can be used to differentiate between poses. Finally the hand is classified in the different poses that we want to recognize.

This approach is based on the use of a statistical learner, a classifier. A classifier is an algorithm that can learn to differentiate between input samples based on: annotated samples known as the training set, in this case we call this a supervised classifier, a measure of similarity between samples but without being annotated, in this case we call this an unsupervised classifier. Our system uses a supervised classifier called Support Vector Machine (SVM).

Given that we are going to use supervised learning we need annotated samples to train the classifier. The set of annotated samples forms a dataset. Part of the dataset is used to train the SVM and is called the training set. The other part is used to test the performance and is called the test set.

The training and testing procedures are offline methods in the sense that there is no real time stream of input data to the system but instead is the dataset instances what are used. Once the system is trained and tested then we will use the trained system in an online schema connected to real time inputs. Consequently we can divide the system in two parts: the batch pipeline and the real time

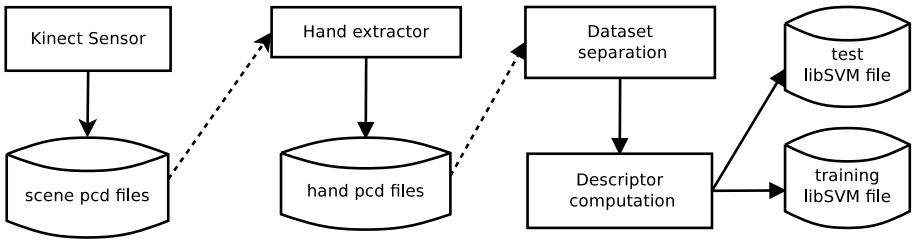


Figure 2.1: Batch pipeline (training).

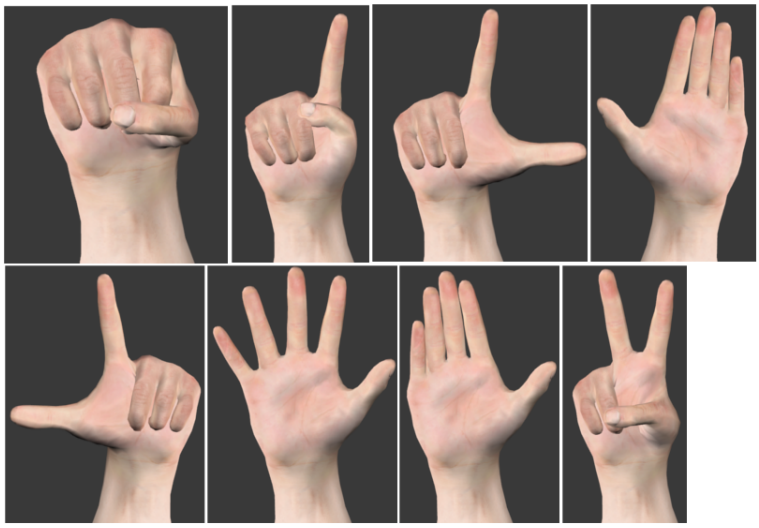


Figure 2.2: Hand classes.

pipeline.

The batch pipeline is used to generate a dataset, train the svm, and test the system, as shown in fig. 2.1. In the batch pipeline first the Kinect sensor captures a frame. We use the Point Cloud Library (PCL) [16] to capture and process the frames. PCL is an open source C++ library that provides with an API for pointcloud processing. The PCL have bundled a kinect driver the OpenNI grabber framework, that we use to obtain 3D pointclouds that have 3D and rgb data. This pointclouds are saved to disk in pcd format files, which is the standard format when working with pointclouds in PCL. This library include utilities to save and load those files.

The files are loaded by the Hand extractor module, which is explained in detailed in section 3. The output of this module is again a pcd file but this time only containing the hand pointclouds and not the hole scene. After that each of the files are selected for training or test with a certain probability (in our experiments we set the probability of belonging to training to 0.75). Then the descriptor module computes a feature vector that is stored in a file in libSVM format. LibSVM [17] is the library that we use that implement SVMs efficiently.

We have recorded a dataset of 9 classes, 8 of them correspond to the hand poses that appear in fig. 2.2. The other class represent the rest of all the possible inputs. In figure 2.4 we can see an example of the dataset. We have stored 700 samples per class. Every sample is a hand pointcloud that contains the coordinates position of every point and its rgb values. The scene pointclouds that the kinect sensor produce contain 640*480=307200 points. However this number is reduced by the detector. The average hand pointcloud size in the dataset have 6437 points.

The SVM is trained using libSVM with the training data. The result of that training is a model file containing the support vectors. With the support vector machine trained, the next thing to do is to test that training. We use the separated instances that are stored in the test file to obtain an accuracy

score. The results are presented in section 6.

The real time pipeline is shown in fig. 2.3. The main differences between the batch pipeline and the real time one is that the real time pipeline does not rely on saving the different outputs in files in the intermediate steps. Also note that the trained support vectors obtained with the batch pipeline are employed here for the classification. The last blocks of the pipeline are optional. The visualization block shows the results in real time, placing a blue sphere in the descriptor position and printing the name of the class recognized by the SVM. The Oblong gripes block transforms the recognized class in a format that Oblong applications can understand, and sends it to a pool. The Oblong particularities are explained in more detail in appendix A.

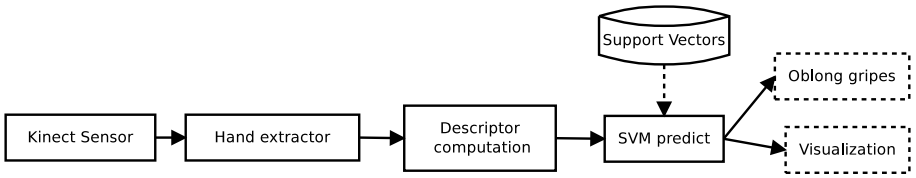


Figure 2.3: Real time pipeline.

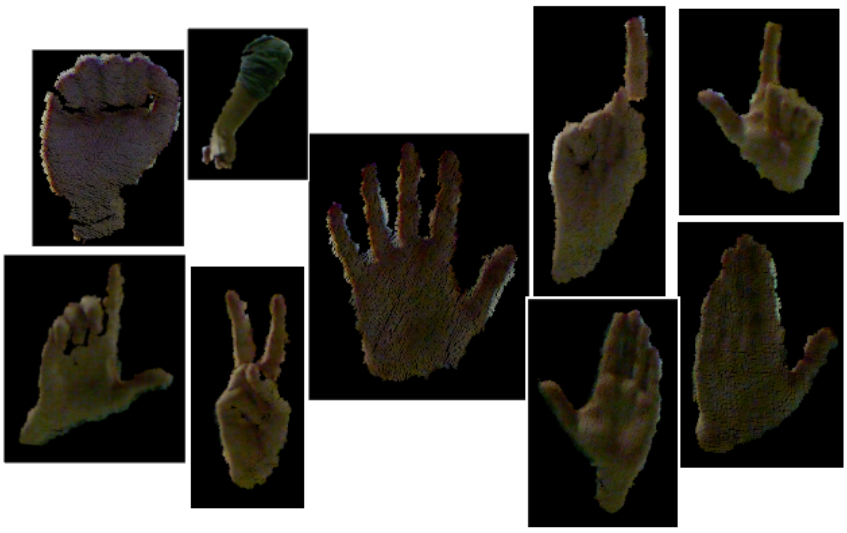


Figure 2.4: Samples from the dataset. One picture per class.

Chapter 3

Detector

The detector module have the task of segmenting the hand from the input pointcloud if it exist. We designed an heuristic that works well in most situations. When a person is performing gestures the most common position for the hands is to be in front of the body. We focus only on gestures that are perpendicular to the viewpoint of the camera, this way the surface that the hand offers to the camera is maximal, and therefore these gestures are easier to recognize with depth sensors.

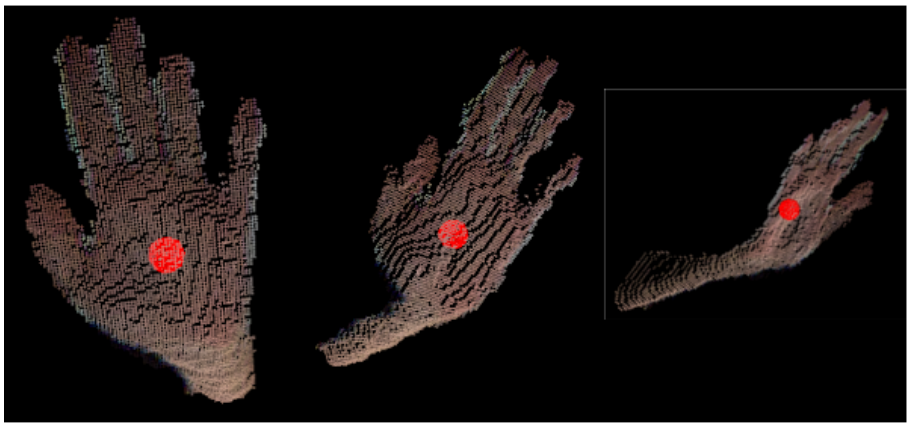


Figure 3.1: Detected hand.

The hand detection algorithm is presented in Algorithm 1. The input of the algorithm is a pointcloud and two thresholds, one in the z component and the other in the distance. A pointcloud in this context mean a set of three dimensional points, the rgb data is not used. First the nearest point to the camera is obtained, this point is called the frontpoint. The hand points should have a z component that lies between the frontpoint z component, and a threshold. If a point is between the frontpoint z component and a threshold we check the distance from this point to the frontpoint. If this distance is lower than a threshold we add this point to the Hand pointcloud. In addition we calculate C, the mean point of the

Algorithm 1 Detector algorithm

Require: a pointcloud I (with n points), a threshold T, a z threshold TZ.
Ensure: a point cloud H (with m points, $m \leq n$), a center point C.

$smallz \leftarrow I_z^{(0)}$
for each point $\alpha \in I$ **do**
  if $\alpha_z < smallz$ **then**
    $smallz \leftarrow \alpha_z$
    $frontpoint \leftarrow \alpha$
  end if
end for
$zthreshold \leftarrow smallz + TZ$
$accx \leftarrow 0$, $accy \leftarrow 0$, $accz \leftarrow 0$, $pcount \leftarrow 0$
for each point $b \in I$ **do**
  if $b_z < zthreshold$ **then**
    $d \leftarrow \sqrt{(b_x - frontpoint_x)^2 + (b_y - frontpoint_y)^2 + (b_z - frontpoint_z)^2}$
    if $d < T$ **then**
      Add b to H
      $accx \leftarrow accx + b_x$, $accy \leftarrow accy + b_y$, $accz \leftarrow accz + b_z$
      $pcount \leftarrow pcount + 1$
    end if
  end if
end for
$C_x \leftarrow (accx/pcount), C_y \leftarrow (accy/pcount), C_z \leftarrow (accz/pcount)$

Hand pointcloud.

This system is very effective but have some limitations, it can only detect one hand at a time, and leave the responsability of checking whether the detection is really a hand or not to the subsequent steps. This works for some applications where the kinect is pointing to a clear area where the only possible object is a person performing gestures in front of the camera. But fails in other settings where the kinect is placed in a busy area where non hands objects can appear and disappear.

Chapter 4

SBSM descriptor

We have designed a descriptor called Spherical Blurred Shape Model (SBSM), that is inspired in [18][15]. The descriptors can be classified in local or global. A local descriptor divides the input in a series of elements that are encoded using the same algorithm. In contrast global descriptors only have meaning if applyied to the whole input. SBSM is a global descriptor that describes the shape of an input 3D pointcloud. The description of the shape is achieved decomposing the input pointcloud in bins defined by an spherical grid.

By defining a spherical grid around the center of the input cloud, the spatial arrangement of object parts is shared among the regions defined by concentric spheres and angular sectors. Additionaly we propose a method to achieve a rotation invariant descriptor by rotating the grid according to the predominant region density. We divide the description of the algorithm into three main steps: the definition of the grid, the descriptor computation, and the rotation invariant procedure.

4.1 Grid definition

Given a number of layers N_l, a number of angular divisions N_θ, a radius R, a center point C, and a 3D pointcloud I, a grid G is defined as a division of the sphere of radius R centered in C in spherical coordinates (see fig. 4.1). The number of layers N_l indicates the divisions in the r component, the number of angular divisions N_θ indicates the divisions in θ and in φ. The result is a series of concentric spheres with increasing radius and angular subdivisions. For example if we have a grid with 6 layers and 13 angular divisions the result will look like figure 4.2. Note that since $\theta \in [0, \pi]$, and $\varphi \in [0, 2\pi]$ the angular distance between adjacent bins is $\frac{\pi}{N_\theta}$ in θ and $\frac{2\pi}{N_\theta}$ in φ.

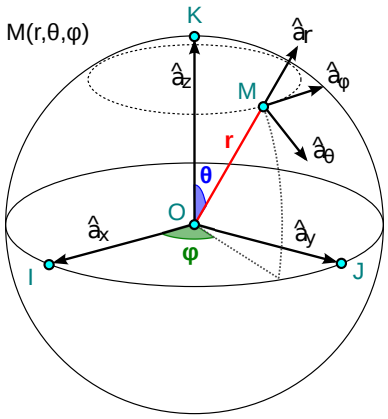


Figure 4.1: Spherical coordinates.

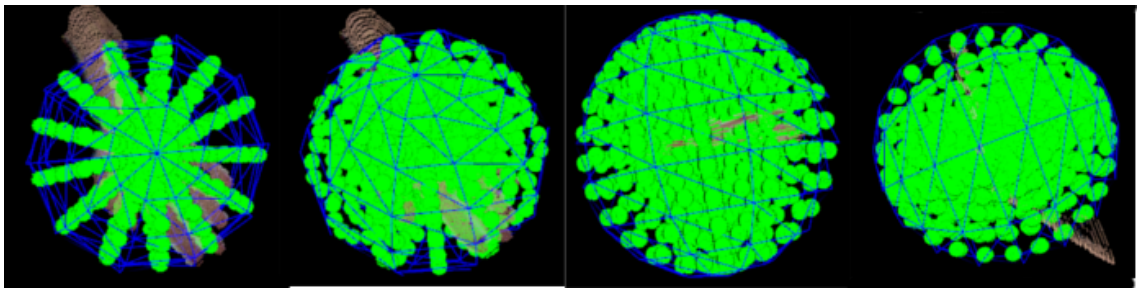


Figure 4.2: Spherical grid with 6 layers and 13 angular divisions. The blue sphere represents the SBSM descriptor. The green spheres represents the bins positions.

4.2 Descriptor Computation

The descriptor procedure is detailed in Algorithm 2. The input cloud I is translated to the center of the descriptor C. The spherical grid is implicitly defined by a vector of centroids. Each centroid is calculated in spherical coordinates and stored it in cartesian coordinates. Every point in the input cloud I, that lies inside the descriptor sphere, is assigned to a bin, incrementing their value. The neighboring bins of each point are increased by a factor that depend on the distance from the point to their centroids. This propagates the weights of a bin to its neightbors smoothing the differences between adjacent bins and making the description tolerant to irregular deformations. Finally the descriptor is normalized dividing each bin by the number of points n. Regarding the computational complexity, note that, for an input point cloud I with n points only $O(n)$ operations are required.

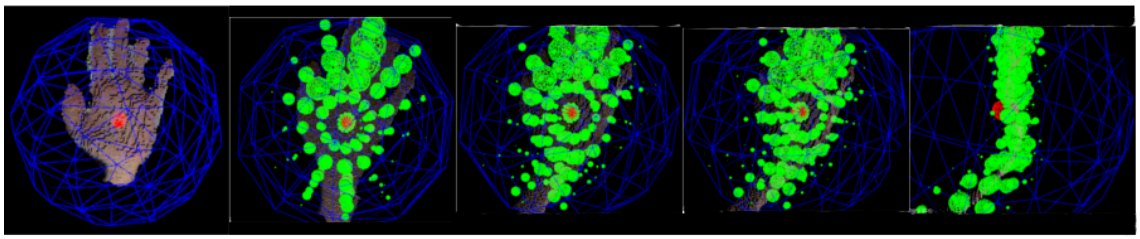
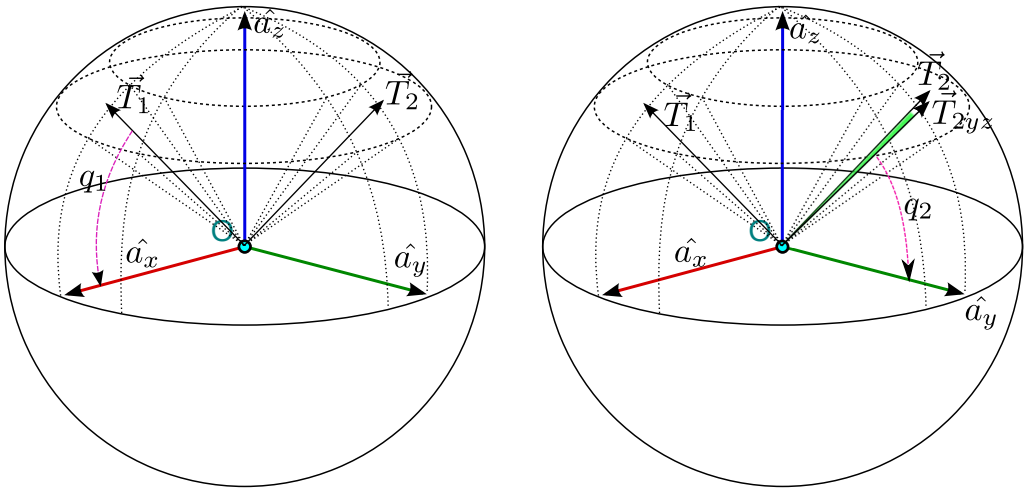


Figure 4.3: SBSM descriptor. The descriptor is represented as a blue sphere. The center of the descriptor is marked by a red sphere. The bins of the descriptor are shown as green spheres with a radius proportional to the bins values.



(a) Quaternion q_1. (b) Quaternion q_2.

Figure 4.4: Quaternions calculation. $\vec{T_1}$ and $\vec{T_2}$ are the principal components of the descriptor.

4.3 Rotation invariant descriptor

In order to make the descriptor invariant to rotation we use a second procedure in the SBSM algorithm. We look for the radial sector T_1 that maximizes the sum of the descriptor values. And for T_2 that is the second largest radial sector. These sectors are then taken as a reference for rotating the descriptor. Given that we are performing a rotation in a three dimensional space, and that we want the alignment of two spheres, we need two quaternions to store it. This operation can be viewed as the transformation of three orthogonal vectors that form a basis. We use the principal axis of the descriptor T_1 and the X axis to compute the first quaternion (see fig. 4.4a). Then the secondary axis of the descriptor T_2 is used. The projection of T_2 in the Y-Z plane and the Y axis are used to compute the second quaternion (see fig. 4.4b). The rotation of the descriptor is achieved using the quaternions to rotate the bins of the descriptor. The details can be viewed in Algorithm 3.

Algorithm 2 Descriptor algorithm

Require: a pointcloud I (with n points), a radius R, a number of angular divisions N_θ, a center point C, and a number of layers N_l.

Ensure: the descriptor vector v.

1: Translate the origin of I to C: $I^{(x)} \leftarrow I^{(x)} - C, \forall x \in [1, n]$

2: $\delta_\varphi \leftarrow 2\pi/N_\theta, \delta_\theta \leftarrow \pi/N_\theta, \delta_r \leftarrow R/N_l$, as the step in each of the spheric components

3: Compute the grid centroids: $B = \{b_{(1,1,1)}, ..., b_{(N_l,N_\theta,N_\theta)}\}$ such that $b_{(i,j,k)}$ is the center of a bin of B between distances $[(i-1) * \delta_r, i * \delta_r]$ to the origin of coordinates and between interval angles $[(j-1) * \delta_\theta, j * \delta_\theta]$ in θ, and between angles $[(k-1) * \delta_\varphi, k * \delta_\varphi]$ in φ

4: initialize the descriptor with zeros: $v^{(i,j,k)} \leftarrow 0, \forall i \in [1, N_l], j \in [1, N_\theta], k \in [1, N_\theta]$

5: **for** each point $\alpha \in I$ **do**

6: transform α to spherical coordinates: $r_\alpha = \sqrt{\alpha_x^2 + \alpha_y^2 + \alpha_z^2}$,

$$\theta_\alpha = \begin{cases} \arctan(\frac{\sqrt{\alpha_x^2+\alpha_y^2}}{\alpha_z}) & \alpha_z > 0 \\ \frac{\pi}{2} & \alpha_z = 0 \\ \pi + \arctan(\frac{\sqrt{\alpha_x^2+\alpha_y^2}}{\alpha_z}) & \alpha_z < 0 \end{cases}, \varphi_\alpha = \begin{cases} \arctan(\frac{\alpha_y}{\alpha_x}) & \alpha_x > 0, \alpha_y > 0 \\ 2\pi + \arctan(\frac{\alpha_y}{\alpha_x}) & \alpha_x > 0, \alpha_y < 0 \\ \frac{\pi}{2} sgn(\alpha_y) & \alpha_x = 0 \\ \pi + \arctan(\frac{\alpha_y}{\alpha_x}) & \alpha_x < 0 \end{cases}$$

7: calculate the bin that correspond to α: $bin_r = \lfloor \frac{r_\alpha}{\delta_r} \rfloor$,$bin_\theta = \lfloor \frac{\theta_\alpha}{\delta_\theta} \rfloor$, $bin_\varphi = \lfloor \frac{\varphi_\alpha}{\delta_\varphi} \rfloor$

8: increment the descriptor bin: $v^{(bin_r, bin_\theta, bin_\varphi)} \leftarrow v^{(bin_r, bin_\theta, bin_\varphi)} + 1$

9: **for** each $b_i \in Neighbors(b_{(bin_r, bin_\theta, bin_\varphi)})$ **do**

10: distance between the point and the centroid: $d \leftarrow |\alpha - b_i|^2$

11: update the descriptor bins: $v^{b_i} \leftarrow v^{b_i} + e^{-d/R}$

12: **end for**

13: **end for**

14: **for** each point $\beta \in v$ **do**

15: normalize the descriptor: $\beta \leftarrow \frac{\beta}{n}$

16: **end for**

Algorithm 3 Rotation invariance algorithm

Require: the descriptor vector v, a number of angular divisions N_θ, a number of layers N_l, and the grid centroids B.

Ensure: the rotation invariant descriptor vector v.

1: Compute the principal orientations: $f(\theta,\varphi) = \sum_{r=1}^{N_l} v^{(bin_r, bin_\theta, bin_\varphi)}$, $T_1 = \underset{\theta,\varphi}{\operatorname{argmax}}\, f(\theta,\varphi)$, $T_2 = \{\theta'', \varphi'' | \{\forall\theta', \varphi'\} - T_1 : f(\theta',\varphi') \leq f(\theta,\varphi), f(\theta,\varphi) \leq f(T_1)\}$

2: Compute quaternion q_1: $\hat{a}_x = (1,0,0)$, compute $\vec{T_1}$ the orientation vector in cartesian coordinates, q_1 is computed from $\vec{T_1}$ to $\hat{a}_x$

3: Compute quaternion q_2: $\hat{a}_y = (0,1,0)$, compute $\vec{T_2}$ the orientation vector in cartesian coordinates, $\vec{T_2}yz = \vec{T_2} * \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, q_2 is computed from $\vec{T_2}yz$ to $\hat{a}_y$

4: **for** every $v^{(bin_r, bin_\theta, bin_\varphi)} \in v$ **do**

5: Compute the rotated bin: $b_{rot} \leftarrow q_2 * q_1 * b_{(bin_r, bin_\theta, bin_\varphi)}$

6: Assign the rotated bin: $v^{b_{rot}} \leftarrow v^b$

7: **end for**

Chapter 5

Classification

Our system uses a Support Vector Machine (SVM) [19], a discriminative classifier that has proved to be one of the most robust classifiers. We rely on the libSVM implementation [17] of SVM to classify our input instances. In our case the input instances are hands shapes described by the proposed SBSM descriptor.

Given that we want to recognize between several hand poses our problem is a multiclass classification problem. SVMs are binary classifiers, therefore they are only able to solve two class problems. Nevertheless the multiclass problem can be solved combining SVMs. "One vs. all" and "one vs. one" are two different ways of combining binary classifiers to solve a multiclass problem. If n is the number of classes, "one vs. all" works dividing the problem in n subproblems where each class is classified against the rest. The class that obtains the highest output function is considered to be the class prediction. On the other hand "one vs. one" groups the classes in pairs and classify each pair separately. The number of classifiers needed for one vs. one is $K = \frac{n(n-1)}{2}$. Every classifier makes a vote on one class, and the class with most votes is assigned as the predicted class. In our case we use libSVM that implements "one vs. one" to solve the multiclass problem.

All the SVMs work maximizing the margin that separates the samples from different classes. However there exists different SVM formulations for classification. In this work we use the C-SVM formulation. The C-SVM formulation considers the following problem. Given training vectors $\mathbf{x}_i \in R^n, i = 1, ..., l$, in two classes, and an indicator vector $\mathbf{y} \in R^l$ such that $y_i \in \{1, -1\}$,

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \tag{5.1}$$

$$\text{subject to } y_i(\mathbf{w}^T\phi(x_i) + b) \geq 1 - \xi_i,$$
$$\xi_i \geq 0, i = 1, ..., l,$$

where $\phi(\mathbf{x}_i)$ maps $\mathbf{x}_i$ into a higher-dimensional space and $C > 0$ is the regularization parameter.
Another important choice when using SVMs is the kernel. The kernel is the $\phi(\mathbf{x}_i)$ function that maps

the input feature vector to a higher dimensional space. Some common choices for SVM kernels are: linear, polynomial, radial basis function and sigmoid. The linear kernel is the simplest kernel and the fastest to compute yet it can only classify according to a linear separability of the instances and when the instances are non linearly separable it perform worse. The radial basis function is more computational expensive but can deal with linear and non linear separability.

The Radial Basis Function has the form $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$. Note that the γ parameter needs to be set before we can train the SVM. Using a C-SVM formulation with a Radial Basis Function as the kernel we have to configured two parameters before training: the C and the γ parameters. These are called the SVM hyperparameters.

Chapter 6

Experiments

In this chapter we explain the experiments performed. First we study how the SBSM parameters affects the overall accuracy. Then we analyze the procedure made to improve the selection of the SVM hyperparameters. Then we examine the effect of the neighbors propagation in the SBSM descriptor. Then we test the computational complexity of each part of the real time system. And finally we show an example of the integration of our system with the Oblong stack.

All the experiments have been made with the following experimental setups:

- The hand extractor module threshold was set to 9 cm with a trial and error procedure.
- The dataset was splited randomly between training (75%) and test (25%).
- The radius of the SBSM descriptor was set to 17 cm.
- The rotation invariant procedure of SBSM was not used.
- We use a Radial Basis Function(RBF) kernel for the SVM.
- The SVM hyperparameters (C, γ) were set through 5 fold cross-validation on the training set.
- Every experiment was repeated 5 times with different random separations between training and test.

6.1 Influence of the number of SBSM grid divisions

The SBSM descriptor parameters (N_l, N_θ) were selected so that they maximize the classification accuracy. We have tried all the possible combinations between the number of layers ($N_l \in [1, 6]$) and the number of angular divisions ($N_\theta \in [4, 36]$).

To select the SVM hyperparameters we use a greedy algorithm instead of exploring all the possible values. This greedy algorithm climbs the space formed by the parameters C and γ and stops when it reaches a 92% value of cross validation accuracy. Then the SVMs are trained and tested. The testing

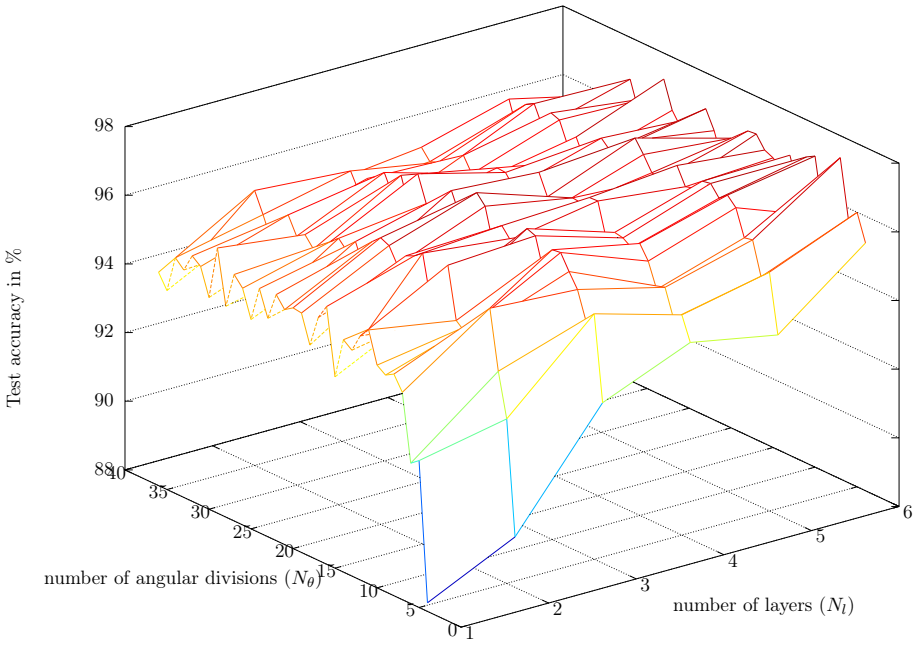


Figure 6.1: System accuracy as a function of the number of layers (N_l) and the number of angular divisions (N_θ).

procedure is the result of applying the trained SVM to the test samples. We use the accuracy of the testing procedure as a performance measure for the classifier.

The experiment results are shown in figure 6.1. The purpose of this experiment was to study the relation between the SBSM parameters and the accuracy of the descriptor. Our results show that even for small N_l and N_θ we can achieve high accuracy values. Also the figure 6.1 shows that the accuracy for a sufficiently big N_θ ($N_\theta \in [15, 40]$) only depends on the N_t values and that even this variation is small. Note that since we stop the cross-validation procedure to select the SVM hyperparameters when it reachs the 92% we can not assure that the accuracy value for a given N_l and N_θ is the maximum. Instead the figure expresses a rough approximation of that maximum.

In the following we perform experiments with the SBSM parameters fixed to $N_l = 6$ and $N_\theta = 13$. These values were selected because they gave us a good balance between high acuracy and constrained length of the descriptor. The length of the descriptor depends on the SBSM parameters as: $length_{SBSM} = N_l * N_\theta^2$. And this length has an impact on the complexity of the classifier. So a smaller length is desirable.

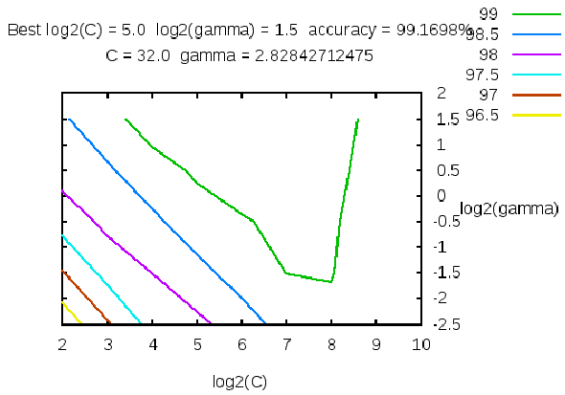


Figure 6.2: Hyperparameter selection using five fold cross validation and an exhaustive search.

6.2 Hyperparameter selection with cross validation

The next experiment that we perform was a fine tune of the SVM hyperparameters. We saw previously that we have used a greedy algorithm to climb the hyperparameters space. This time we explore in detail such space, maximizing the cross validation accuracy (see fig. 6.2). Setting $C = 2^5 = 32$, $\gamma = 2^{1.5} = 2.82$, $N_l = 6$, and $N_\theta = 13$ we obtain a classification accuracy of 99.28% (1517/1528) with a feature vector of dimension $N_l * N_\theta^2 = 1014$. These fine tuned parameters are used to train the classifier that will be used in the real time pipeline.

6.3 The effect of the neighbors propagation in SBSM

We wanted to study the effect that the neighbors propagation of SBSM (see algorithm 2) has on the accuracy of the shape description. Hence we have compared the accuracy of our system using SBSM and a simplified version of SBSM that do not propagate weights to the neighboring bins. We called this simplified version of SBSM a "Spherical Voxel Grid" (SVG), which is an extension of the standard 2D image zonning descriptor used in many computer vision systems. To be a fair comparison we have used the same experimetal setup for both approaches. The accuracy using the SBSM descriptor was 99.28%, using the Spherical Voxel Grid the accuracy was 98.3376%. The small difference between the two tell us that for this application the increment in accuracy do not worth the additional cost of using the SBSM descriptor. However in other applications where the objects to be recognized have an irregular shape the neighbor propagation can have a different effect in the system accuracy.

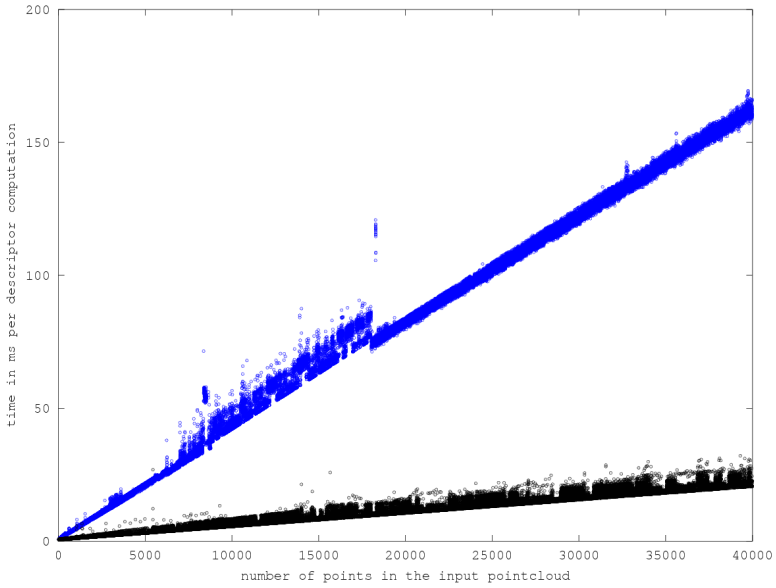


Figure 6.3: Time spent (ms) in the computation of the descriptors as a function of the number of input points n. In blue the SBSM descriptor, in black the Spherical Voxel Grid descriptor.

6.4 Computational cost

Given the importance of the Human Computer Interaction applications and the fact that these applications require low latency algorithms to be feasible, it is important to study the computational cost of our algorithms. We study the computational cost of the elements that constitute the real-time pipeline (see fig. 2.3): hand detector, hand descriptor, and classifier. All the experiments were done in a Intel(R) Core(TM) i5-2430M CPU @ 2.40GHz with 4Gb of RAM laptop. The program runs in a single thread of execution.

6.4.1 Descriptor

We analyze the computational cost of the SBSM descriptor and the Spherical Voxel Grid (SVG). The computational cost of both algorithms depends on the number of points n of the input pointcloud I. Both algorithms realize $O(n)$ operations per input. However the SBSM descriptor does more work per point. In figure 6.3 we can see how the time spent in the descriptor grows with n. The SVG descriptor is faster than SBSM and we can see how this difference grows with n. Also note that the computation time grows linearly with n. To perform this experiment both descriptors were set to the same parameters:

- Both descriptors uses $N_l = 6, N_\theta = 13$.
- The points of I are synthetic points with a random position inside the descriptor sphere.

In figure 6.4 we can see how the framerate drops faster in the case of the SBSM descriptor that in the Spherical Voxel Grid. However the SBSM descriptor maintain acceptable framerates at 5000 points. In

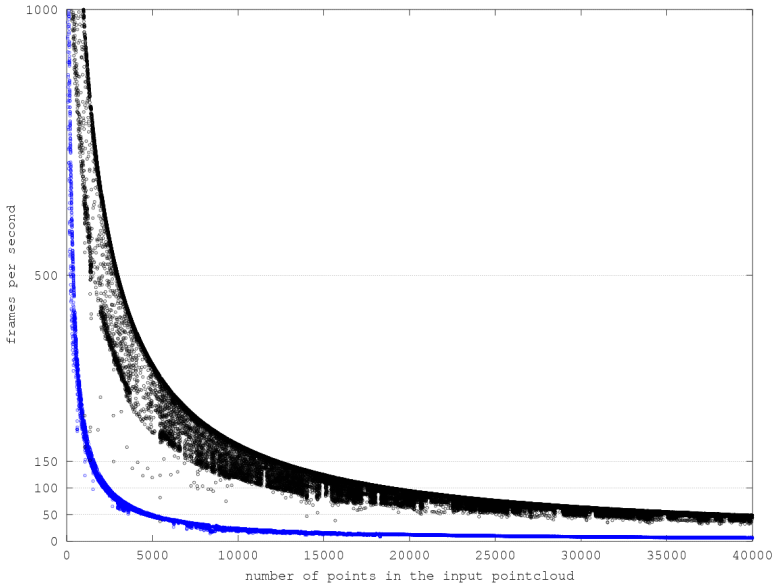


Figure 6.4: Frames per second as a function of the number of input points n. In blue the SBSM descriptor, in black the Spherical Voxel Grid descriptor.

our dataset the average number of points is 6437. So SBSM has an acceptable framerate in our case.

6.4.2 Detector

To study the computational cost of the detector we record the time spent in the detector module while performing gestures in front of the kinect. We record 180 samples and calculate the mean and the standard deviation. The average time spent in the detector was 14.590 ± 1.2940 ms, and the average framerate was 68.949 ± 4.7038 fps.

6.4.3 Classifier

We study the computational cost of our classifier in the context of our real time system. The time spent by the classifier was measure while performing gestures in front of the kinect device. We record 319 samples and calculate the mean and the standard deviation. The average computational cost of the classification was 5.2635 ± 0.63316 ms, and the average framerate was 191.26 ± 18.867 fps.

6.5 Oblong integration

We have integrated our hand pose recognition system within the Oblong stack. Our application sends data to a "pool", a communication channel abstraction, in a convenient format. To do that we make use of the Oblong "plasma" library for the interprocess communication. This library handles the transmission of messages called "proteins". In our case those messages contain the hand poses that

our system recognize. Oblong encodes the hand configurations with a symbolic structure called "gripe" (gripe is the acronym for gestural reduced instruction pose expression). Those messages are received by the applications that are connected to the pool, that then can trigger events based on the recognized hand poses. In figure 6.5 we can see an example Oblong application receiving data from our system.

Figure 6.5: Our application in the left, the Oblong application in the right.

Chapter 7

Conclusions

Depth cameras can have a great impact on what is possible in Computer Vision problems by reducing the ambiguities inherent to the 2D representation of a 3D world. Using a kinect camera we have developed a real time Hand Pose Recognition system capable of recognize eight hand poses. This system uses an object recognition approach to recognize the different poses. Following this approach the input depth frames are first processed to extract the hands, then described and finally classified. We have recorded a dataset with nine classes, representing eight different hand poses and a class representing negative samples from the non-considered eight hand configurations. We have used this dataset to train a multiclass SVM classifier. Our results show accuracy values above the 99% it the designed dataset.

For the descriptor part we have presented a novel shape descriptor for 3D data, the Spherical Blurred Shape Model (SBSM), that segments the input pointcloud in zones using a spherical grid. This descriptor copes with the irregular deformations by propagating the weights of a point to its neighboring bins. Furthermore a technique to make this descriptor invariant to rotation was presented. We have studied the influence of the parameters of SBSM on the accuracy of the system and also the effect of the neighbors propagation in SBSM. The study of the computational cost of SBSM shows that SBSM performs well under our average input pointcloud, allowing for real time applications.

Finally our real time system was integrated with the Oblong platform allowing to use their applications without additional wrappings.

As future work we will extend the dataset to been able to recognize more poses. Also we will compare the SBSM descriptor with other 3D descriptors, and analyze how it performs with different types of data. Moreover we plan to assess the performance of the rotation invariant procedure of SBSM. Regarding the detector we want to further explore and analyze other alternatives to deal with more complex scenarios.

Appendix A

Oblong concepts

Oblong industries is a company dedicated to, between other topics, Human Computer Interaction. They build systems that simplify the way computers are controlled, allowing for novel interactions between man and machine and expanding the notion of interfaces. To support this vision they have software that can handles not only the incoming commands but the radical new ways in which information can be displayed.

The Spatial Operating Environment (SOE) is a key concept that they developed in which the space that surrounds the system has an explicit representation. This explicit representation allows among other things to raise the level of abstraction in display technologies. For example in a setting with multiple screens the average computer will treat them as separate entities and this difficults the programming of multiscreen applications. In SOE applications all you need is a configuration file telling the system where in the space are those screens and the orientations of each of them. The underlying software will take care of the details of how to make the screens work, liberating the application programmer from that task. This is particularly important in complex settings where you do not only have multiple screens but also projectors and surfaces. Summarizing, computers communicate with humans through interfaces, SOE generalizes the interfaces of a computer, embedding them in an spatial representation of the surrounding world. This is not only true with the displays but also with the interaction devices that "live" in the space.

G-Speak is the SDK that implements all of this functionality. In the moment of writing this report there exist versions of this SDK in C/C++, Java and Ruby languages. G-Speak is compose of different libraries that used in cooperation can help building complex applications. Without pretending to be exhaustive the functionality of G-Speak covers:

- data structures and programming abstractions.
- networking, interprocess communication.
- events, time management.

- video and audio.
- UI capabilities, graphical interfaces, gesture support.

G-speak uses a biologicaly inspired terminology for referring to its constructs. As for example the library that implements interprocess communication is called "plasma". The messages that are send between processes are "proteins" and the channel to communicate those proteins is called a "pool".

Another important concept related to gesture representation is the concept of gripe (short for gestural reduced instruction pose expression). A gripe is the representation of a hand pose in a symbolic structure. Its a seven length string of characters divided in two groups: the first five characters describe the individual finger positions, and the last two describe the palm and finger orientation. These two groups are separated by a colon.

The finger position characters are ordered pinky, ring, middle, index and thumb. Each of them can have values: "^" curled, "|" straight, and "x" palm normal. Except the thumb that has its own values: "|" straight, ">" curled thumb and "-" out.

The palm and finger orientation can have six different values:

- Cranial: toward the head or up "^".
- Caudal: toward the ground, or down "v".
- Anterior: in front of you, or away from the body "x".
- Posterior: behind you or toward the body ".".
- Medial: toward the middle of the body "-".
- Lateral: away from the middle of the body "+".

In figure A.1 an example of the gripes representation.

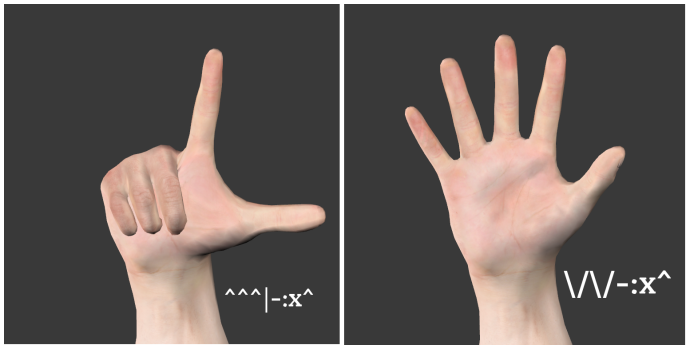


Figure A.1: Hand poses with gripes representation in white.

Bibliography

[1] N. Vo, Q. Tran, T. B. Dinh, T. B. Dinh, and Q. Nguyen, "An efficient human-computer interaction framework using skin color tracking and gesture recognition," in *Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2010 IEEE RIVF International Conference on*, nov. 2010, pp. 1 –6.

[2] V. Athitsos and S. Sclaroff, "Estimating 3d hand pose from a cluttered image," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 2, june 2003, pp. II – 432–9 vol.2.

[3] B. Stenger, "Template-based hand pose recognition using multiple cues," in *In Asian Conference on Computer Vision*, 2006, pp. 551–560.

[4] R. Rosales, V. Athitsos, L. Sigal, and S. Sclaroff, "3d hand pose reconstruction using specialized mappings," in *In ICCV*, 2001, pp. 378–385.

[5] R. Rusu, Z. Marton, N. Blodow, and M. Beetz, "Learning informative point classes for the acquisition of object model maps," in *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, dec. 2008, pp. 643 –650.

[6] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ser. ICRA'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 1848–1853. [Online]. Available: http://dl.acm.org/citation.cfm?id=1703435.1703733

[7] M. Alexa, S. Rusinkiewicz, M. Alexa, and A. Adamson, "On normals and projection operators for surfaces defined by point sets," in *In Eurographics Symp. on Point-Based Graphics*, 2004, pp. 149–155.

[8] N. J. Mitra, A. Nguyen, and L. Guibas, "Estimating surface normals in noisy point cloud data," in *special issue of International Journal of Computational Geometry and Applications*, vol. 14, no. 4–5, 2004, pp. 261–276.

[9] G. Burel and H. Henoco, "Determination of the orientation of 3d objects using spherical harmonics," *Graph. Models Image Process.*, vol. 57, no. 5, pp. 400–408, Sep. 1995. [Online]. Available: http://dx.doi.org/10.1006/gmip.1995.1034

[10] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz, "Rotation invariant spherical harmonic representation of 3d shape descriptors," in *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, ser. SGP '03. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 156–164. [Online]. Available: http://dl.acm.org/citation.cfm?id=882370.882392

[11] D. Saupe and D. V. Vrani, "3d model retrieval with spherical harmonics and moments," in *DAGM*. Springer-Verlag, 2001, pp. 392–397.

[12] M. Ben-Chen and C. Gotsman, "Characterizing shape using conformal factors," in *3DOR'08*, 2008, pp. 1–8.

[13] R. Rusu, G. Bradski, R. Thibaux, and J. Hsu, "Fast 3d recognition and pose using the viewpoint feature histogram," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, oct. 2010, pp. 2155 –2162.

[14] D. Minnen and Z. Zafrulla, "Towards robust cross-user hand tracking and shape recognition," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, nov. 2011, pp. 1235 –1241.

[15] S. Escalera, A. Fornés ands, O. Pujol, J. Lladó ands, and P. Radeva, "Circular blurred shape model for multiclass symbol recognition," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 41, no. 2, pp. 497 –506, april 2011.

[16] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[17] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[18] S. Escalera, A. Fornés, O. Pujol, P. Radeva, G. Sánchez, and J. Lladós, "Blurred shape model for binary and grey-level symbol recognition," *Pattern Recognition Letters*, vol. 30, no. 15, pp. 1424 – 1433, 2009.

[19] V. N. Vapnik, *The nature of statistical learning theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.