,

3D Hand and Finger Recognition using Kinect

F. Trapero Cerezo[1]

[1]Universidad de Granada (UGR), Spain

Abstract

Finger and hand tracking are included in the field of the human-computer interaction (HCI). The goal of this work is to describe a robust and efficient method to track the hand and the fingers in real time using the Kinect device. This device allows us to obtain a depth image, instead of a normal RGB image, which gives us more information than a normal camera. These kind of images are more useful for the tracking, and moreover we can represent the relevant points in a 3D space. The final idea is to use the algorithms of this paper to create immersive virtual environments applications.

Categories and Subject Descriptors (according to ACM CCS):
I.4.8 [Image processing and computer vision]: Scene Analysis—Tracking
I.3.7 [Computer graphics]: Three-Dimensional Graphics and Realism—Virtual Reality

1. Introduction

Human-computer interaction (HCI), and in particular hand and finger tracking, depends largely on physical devices. People have worked on this issue for years, but as technology advances, new devices and techniques appears that allow us to accomplish our objective in a more efficient way.

In order to control our computer we use devices like the mouse or the keyboard, but it seems unnatural for humans. We are used to interact with the world with our own hands, our body or our voice. That is the reason that hand and finger tracking has become more popular, these last years.

So far, if we wanted the computer to know where our hands were, we would have to wear some special gloves. This kind of devices are really expensive, so they are only used in big projects of virtual reality. However, technology have advanced really fast, and now computers can analyze images and get the contour of a hand in real time.

Progress in this type of technologies has led to the creation of devices like Kinect. Kinect is a cheap device which appeared in the latest of 2010, that has a lot of potential for HCI applications. This device is compound by a normal camera and two infrared cameras. Using the infrared cameras, Kinect is able to generate a depth image.

This paper contains the methods needed to track several hands and their components (fingertips and palm) in a 3D space, based on the depth image that Kinect (or other device) generates.

1.1. Previous work

Before having devices to get a depth image, algorithms of hand and finger recognition were based on the pigmentation of the hand, as we can see in [Mal03], to recognize the shape of the hand. But these kind of algorithm has an obvious problem, it depends on the hand color. The algorithm can only be used on one kind of pigmentation, light or dark. So, depending on the person, the algorithm must be changed.

Anyway, algorithms created for this kind of hand recognition are still useful for depth images, because once you have the shape of the hand, in both cases, these algorithms can be applied. Many of the algorithms commented in this paper, have been used for hand recognition in normal RGB images.

There are mainly two ways to deal with this problem. The first one is to use a large amount of hands images to train our system and try to find correspondences with them. The second option is to use some geometric properties of our hands, to find out some important information, like where are the fingertips and the center of the hand.

The first approximation is discussed in [SMC01, Che08], and it can provide good results in model-based 3D hand tracking. The main advantage of this method is that we can generate a 3D model of the hand. However, to create this model we need a huge set of hands images to train our system. This is too expensive computationally, so it is really difficult to achieve a real time hand recognition, which is one of our main goals. Besides the efficiency, there are other issues like scalability, it is difficult that the parameters learned during the training, suits to a large variety of hand sizes.

The other option is to calculate the contour and inside points of the hand, and then to look for in this points which are the candidates to be a fingertip or the hand center. This method is more efficient than the previous one, but the generation of a 3D model hand is less accurate and difficult, due to it is only possible to obtain the 3D coordinates of relevant points. However, it provides us with a good approximation of the hand in a really efficient way.

We have chosen the second option because of the efficiency, so that gives us a quick and robust method to track more than one hand in real time.

2. System overview

In order to find the the fingertips and the palm center, the program have to walk trough all these steps.

1. Generate near pixels matrix
2. Decrease noise
3. Classify in contour or inside pixels
4. Differentiate hands and calculate their contour
5. Allocate inside points
6. Find the center of the palm
7. Find the fingertips
8. Allocate points in a 3D space

In this section we are going to describe different ways to achieve these goals, and some modifications of the main algorithm to improve the efficiency of the code.

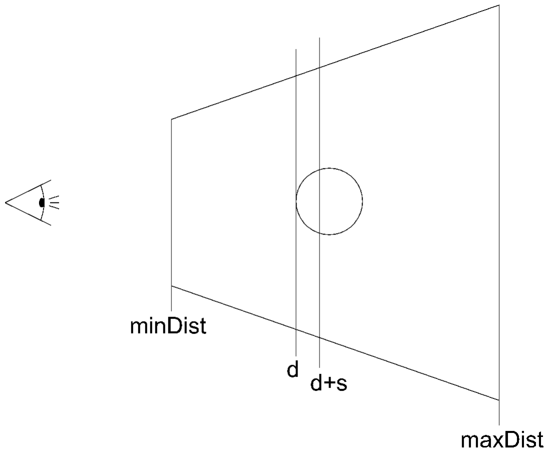
2.1. Generate near pixels matrix

The first step we need to accomplish in order to start to work with the depth data, is to decide which pixels are we going to take into account, to carry on the tracking. The Kinect can catch the distance of the points which are visible to the camera, between the values *minDist* and *maxDist* (Figure 1).

We are going to base our choice using the proximity to the Kinect. We could choose if a pixel is near, by one of this method:

- **Absolute depth**. A pixel is near if its depth is lesser than a constant value, that means that the pixel is between *minDist* and a predefined value which is greater than *minDist* and lesser than *maxDist*.
- **Relative depth**. First of all, we calculate the minimum depth, and in base of that depth we select a maximum depth (for example by adding a constant value to the minimum depth). So, if the depth is between these two values, the pixel is near. This method allow us to have greater mobility, because it does not force us to stay in the same position the whole time. In the Figure 1 the minimum depth is *d* and we add a constant *s* to it, so every point between *minDist* and *d+s* will be accepted as near.

Figure 1: *Relative depth method representation*



After selecting a method, we must select the resolution of the depth image. It is possible to select it from three different options 80x60, 320x240 or 640x480. The best option is obviously the 640x480 resolution, but in order to improve the efficiency of the code, we can choose the 320x240 resolution, because it gives us enough definition to distinguish the contour of the hands. This choice greatly reduce the number of operations, and consequently, it will improve the efficiency of the code.

Once we have the depth data, we should generate a matrix of the corresponding size, which contains if each pixel is close enough or not, following the criterion selected at the beginning. This matrix will let us to access the data in an easier way, due to many of the algorithms need to know the adjacent pixels of a given one.

One final improvement is to set to false the values of the borders of the matrix. This way, we have not to check each time if a matrix access is out of bounds. That will increase the efficiency in the contour algorithm, and we do not lose almost information.

2.2. Decrease noise

This step is optional, but sometimes could help if we are too far from the Kinect or there too much noise. It consists on applying one or two morphological transformations used in artificial vision, dilation and erosion. These techniques are discussed in [MdPAE*06].

The dilation and erosion are used for expanding and contracting the shapes contained in a binary image respectively. In this case, the hands are these shapes. Both methods apply a mask B over a matrix A.

Applying dilation using a circle as the mask will round the contour of the hand, and will decrease the noise. But, if the dilation is too strong we can lose the shape of some fingers, so after the dilation we could apply erosion with the same mask. Dilation is not the opposite operation of erosion, so the final image will contain a rounder shape which preserves the same size of the first one.

These methods are not cost effective, they are only needed in case that the algorithm cannot find the fingertips.

2.3. Classify in contour or inside pixels

Finally we have a binary matrix which contains the shape of the hands. Each pixel in this matrix points out if the pixel is part of the hand (is valid) or not. The valid pixels of this matrix must be divided in contour or interior pixels. A pixel is interior if every adjacent pixel (up, down, left and right pixels) are also valid, and contour if one of its adjacent pixels are not valid.

This information should be stored in two different lists, otherwise we have to go over the whole matrix several times, in order to calculate the sorted contour and the palm center.

2.4. Differentiate hands and calculate their contour

It seems that the contour is already calculated, but we need a sorted contour for calculating the fingertips and to improve the efficiency of the calculation of the center of the palm. A sorted contour means that the contour must be a list of points, in which, the previous and next point of each point in the list are adjacent in the contour of the hand, and the first and last point are also adjacent.

The contour is calculated applying the “turtle algorithm” explained in [MdPAE*06]. The pseudocode of the algorithm is shown in Algorithm1.

Each time a point is added to the contour of one hand, that point is marked as visited. So, after finishing the contour of one hand, the application looks for other contour point to calculate the sorted contour of another hand. If all the contour points has been visited, there are no more hands to analyze. Must take into account that could be some noise that create some small shapes. If the contour is so small compared to the total number of contour points, we must discard it.

Algorithm 1 Turtle algorithm

```
function TURTLEALGORITHM($startPoint$, $contour$, $valid$)
    $currPoint := startPoint$
    $dir := 0$
    $lastPoint := Point(-1,-1)$
    while $True$ do
        if $currPoint \neq lastPoint$ then
            $list.add(currentPoint)$
            $lastPoint := currentPoint$
            $contour[currPoint.Y][currPoint.X] := False$
        else
            $dir := (dir + 4 - 1)$
        end if
      # Change the direction
        if $dir = 0$ then
            $currPoint.X := currPoint.X + 1$
        end if
        if $dir = 0$ then
            $currPoint.Y := currPoint.Y + 1$
        end if
        if $dir = 0$ then
            $currPoint.X := currPoint.X - 1$
        end if
        if $dir = 0$ then
            $currPoint.Y := currPoint.Y - 1$
        end if
      # Finish if comes back to starting point
        if $currPoint = startPoint$ then
            $break$
        end if
    end while
    return $list$
end function
```

2.5. Allocate interior points

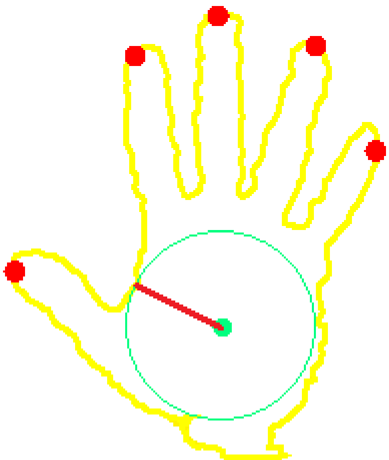
At this point, we have already calculated the number of hands we have, and their contour. The next step is to allocate each interior point into the current hand. One heuristic could be to calculate the container box. The container box of a figure is delimited by two points. These two points delimit the smallest box which contains every pixel of the figure.

We can calculate the container box using the contour points, and with the container box we can classify very quickly if a interior point is inside the container box and therefore in the figure. There are some problems when the container boxes overlap, and the same interior point is inside more than one box. In this case, the pixel is wrongly classified. These kind of errors does not affect the final result, due to the objective of classify the interior points into the correct hand is to find the center of the palm, and the misclassified pixels can never be the center of the palm.

2.6. Find the center of the palm

The center of the palm is one of the points that gives us a lot of information. This point indicates where is the hand, and

Figure 2: *Graphical interpretation of the calculation of the center of the palm*



together with the fingertips, could be used for calculating other relevant information which, for example, could help us to identify gestures.

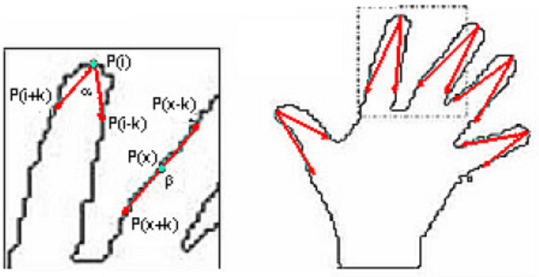
The center of the hand is, usually, the center of the biggest circle that fits inside the contour of the hand. In order to find this point, we calculate the minimum distances from each interior point to every contour point, and from these distances we select the maximum. The point that corresponds to that distance is the center of the hand.

This algorithm consume a lot of time, so we must improve its efficiency. We have done it in two ways:

- While calculating the minimum distance of an interior point to every contour point, if the current minimum is lesser than the current maximum, we can assure that this point is not the center, so there is no need of calculating the distance to the rest of contour points for this interior point.
- Due to in the list of the interior and contour points, consecutive points are near in the hand, we can only do the calculations for 1 in N consecutive points. Using a small value of N greater than 1, the error is negligible and the efficiency is improve by $1/N^2$. Actually, with values around 8 the results are quite acceptable.

In the 2obtained from [Ste] we can observe what are we looking for when we use the algorithm. The circle in the figure is the biggest one that the hand shape can contains, so the center of that circle should be the center, and indeed it is the center. Sometimes we could have problems if the arm is showed, and the algorithm allocate the center in the arm. Therefore we have to be careful and avoid showing the arm as far as possible, by for example delimiting the parameters explained in section 2.1.

Figure 3: *K-curvature Algorithm*



2.7. Find the fingertips

The method used to find the fingertips is the k-curvature algorithm. The main idea is for each point P(i) of the contour, get the P(i-k) and P(i+k) points, and using these points generate two vectors and calculate the minimum angle they form. The vector are formed by P(i-k) - P(i) and P(i+k) - P(i). If the angle is lesser than an α value, it is a fingertip. The more robust values found were k=22 and α = 40.

One of the problems of this algorithm is that it could produce some false positive in the valley between two fingers, because they have similar properties to the fingers. To avoid this false positive, we calculate the distance between the center of the hand an two points, P(i) and the middle point of P(i-k) and P(i+k). If the distance is greater in the first case, it is fingertip, in other case is valley.

Moreover to improve the efficiency we introduce a heuristic. Each time we deduce that P(i) is a fingertip, the next point we should deal with is not P(i+1), it should be P(i+m), because fingertips are not near to each other. The value of m depends on the size of the contour of the hand, around a 10% of the number of points of the contour seems to be reasonable.

Using the Figure 3 (obtained from [TP10]) we can understand more clearly how the algorithm works. We have two cases, the first one try to calculate the angle α that is formed by the lines P(i+k) - P(i) and P(i-k) - P(i). In this case the angle is around 30°, so the point P(i) it could be a fingertip. But, in the second case we calculate the angle β using he lines P(x+k) - P(x) and P(x-k) - P(x), and the resulting angle is equal to 180° more or less, so the point P(x) cannot a fingertip. The value of k in this sample is too small, and we can find more points than we should, so the idea is to find the value k that fits better with the right side of the figure.

2.8. Allocate points in a 3D space

Everything explained above is used for hand and finger recognition in 2D space. But how can we extend these 2D points into 3D? The solution is really simple using Kinect, due to we have the depth of each 2D point, so we have only

Figure 4: *3D Normalization*

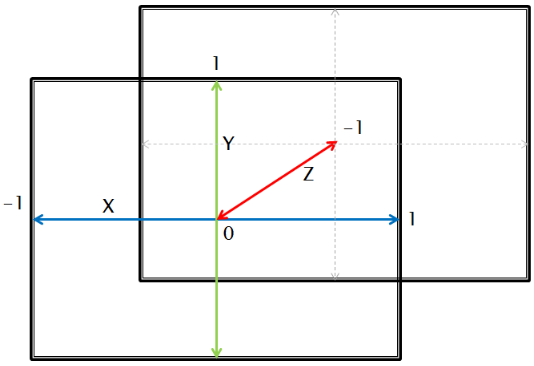
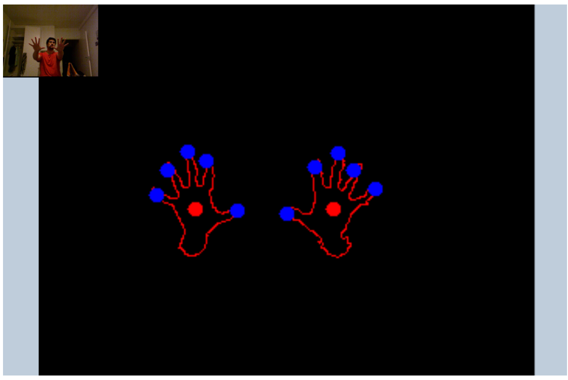


Figure 5: *Test application example*



to normalize the values and add the depth data to each point. This way we have the fingertips and the center of the palm located in a 3D space.

To normalize the points we have decided to use the same normalization that Kinect uses for skeleton tracking, this normalization is shown in the Figure 4. The data we obtain from each point is a X and Y value contained in the box whose left-upper corner is (0, 0), and the right-down corner is (width, height). Width and height values depends on the resolution we have selected in the section 2.1. The Z value is between 400 and 8000, the vision space the Kinect can catch measured in centimeters, although Kinect is not too accurate for points further than 4 meters.

The formulas to do conversions are:

- $X' = \frac{X}{width*2} - 1$
- $Y' = (1 - \frac{Y}{height}) * 2 - 1$
- $Z' = \frac{Z-400}{-7600}$

3. Developed applications

To test the performance of the algorithms and improvements proposed a test application has been developed (Figure 5). This application uses the library we have developed to perform the finger and hand tracking. With the application we can modify the different parameters mentioned in this document, this way we have been able to find the values that have the better equilibrium between performance and accuracy. The library contains the values of the parameters which has provided us the best results and assigns those values by default.

Final results provide us with a frame rate of 30 fps when we show the most two hands at the same time. But this frame rate becomes slower when the algorithm try to perform the finger and hand tracking with big objects. It is not a big issue, due to we want to perform the hand tracking correctly when the hands are showed. We have also achieved a great level of accuracy in finger tracking, but for calculating the palm center, it oscillates around the real center because of the noise.

We have used the library to create a starship simulator. This simulator uses the 3D position of the hands to control the starship in a 3D space. We have designed the control to be as realistic as possible, like if you were holding a steering wheel. You can also shoot missiles opening and closing your hand. The idea is to be able to control any thing in the most natural way.

4. Conclusions

This paper presented a method to track several hands and how to obtain their relevant points (fingertips and palm center), based on some geometric properties. Not only the basic methods have been explained, but also some improvements. Moreover, it has been proved the depth data that the Kinect can provide is useful not only to recognize the shape of the hand, but also to allocate the points in a 3D space. This kind of tools could be the beginning of a new era of HCI, in which we will control our computers using only our hands and our body.

I would like to thank Rubén Aguilar Becerra, because without our discussions about this topic I did not come up with some ideas, and Miguel Ángel Romero Cobos, who has helped with some graphics. And also to Alejandro J. León Salas who has been the teacher responsible of this research.

5. Future work

The next step of this research is to apply it in immersive virtual environments, using other virtual reality devices to increase the immersion of the user. It could be used in games or navigation systems. But in order to achieve it, we need to add more features, like for example gesture recognition [TP10].

References

[Che08] CHEN Q.: *Real-Time Vision-Based Hand Tracking and Gesture Recognition*. PhD thesis, School of Information Technology and Engineering Faculty of Engineering University of Ottawa, 2008.

[GSK*11a] GIRSHICKY R., SHOTTONY J., KOHLIY P., CRIMINISIY A., FITZGIBBONY A.: Efficient regression of general-activity human poses from depth images.

[GSK*11b] GIRSHICKY R., SHOTTONY J., KOHLIY P., CRIMINISIY A., FITZGIBBONY A.: Efficient regression of general-activity human poses from depth images: Supplementary material.

[Mal03] MALIK S.: Real-time hand tracking and finger tracking for interaction. CSC2503F Project Report.

[MdPAE*06] MARCOS A. G., DE PISÓN ASCACÍBAR F. J. M., ESPINOZA A. V. P., ELÍAS F. A., LIMAS M. C., MERÉ J. O., GONZÁLEZ E. V.: *Técnicas y algoritmos básicos de visión artificial443*. Universidad de la Rioja. Servicio de publicaciones, 2006.

[SFC*11a] SHOTTON J., FITZGIBBON A., COOK M., SHARP T., FINOCCHIO M., MOORE R., KIPMAN A., BLAKE A.: Real-time human pose recognition in parts from single depth images.

[SFC*11b] SHOTTON J., FITZGIBBON A., COOK M., SHARP T., FINOCCHIO M., MOORE R., KIPMAN A., BLAKE A.: Real-time human pose recognition in parts from single depth images: Supplementary material.

[SMC01] STENGER B., MENDONÇA P. R. S., CIPOLLA R.: Model-based 3d tracking of an articulated hand.

[Ste] STEFAN: Center of the palm (hand tracking). http://blog.candescent.ch/2011/04/center-of-palm-hand-tracking.html.

[TP10] TRIGO T. R., PELLEGRINO S. R. M.: An analysis of features for hand-gesture classification. *17th International Conference on Systems, Signals and Image Processing* (2010).