Commodifying Pointing in HRI: 
Simple and Fast Pointing Gesture Detection 
from RGB-D Images

by

Bita Azari

B.Sc., Amirkabir University of Technology (Tehran Polytechnic), 2016

Thesis Submitted in Partial Fulfillment of the 
Requirements for the Degree of 
Master of Science 

in the 
School of Computing Science 
Faculty of Applied Sciences

© Bita Azari 2019
SIMON FRASER UNIVERSITY 
Summer 2019

Copyright in this work rests with the author. Please ensure that any reproduction or re-use 
is done in accordance with the relevant national copyright legislation.
# Approval

**Name:** Bita Azari  
**Degree:** Master of Science (Computing Science)  
**Title:** Commodifying Pointing in HRI: Simple and Fast Pointing Gesture Detection from RGB-D Images  
**Examining Committee:**  
- **Chair:** Mo Chen  
  Assistant Professor  
- **Richard T. Vaughan**  
  Senior Supervisor  
  Professor  
- **Angelica Lim**  
  Supervisor  
  Assistant Professor  
- **Yasutaka Furukawa**  
  Internal Examiner  
  Assistant Professor  

**Date Defended:** April 25th 2019
Abstract

We present and characterize a simple method for detecting pointing gestures suitable for human-robot interaction applications using a commodity RGB-D camera. We exploit a previously published state-of-the-art Deep CNN-based detector to find hands and faces in RGB images, then examine the corresponding depth channel pixels to obtain full 3D pointing vectors. We test several methods of estimating the hand end-point of the pointing vector. The system runs at better than 30Hz on commodity hardware, exceeding the frame rate of typical RGB-D sensors. An estimate of the absolute pointing accuracy is found empirically by comparison with ground-truth data from a vicon motion-capture system, and the useful interaction volume established. Finally we show an end-to-end test where a robot estimates where the pointing vector intersects the ground plane, and report the accuracy obtained. We provide source code as a ROS node, with the intention of contributing a commodity implementation of this common component in HRI systems.

Keywords: Robotics; Human-Robot Interaction; Gesture Detection; Pointing Gesture
To Maman, Pedar and Dina
Acknowledgements

I would like to express my sincere gratitude to my senior supervisor Prof. Richard Vaughan for his patience and immense knowledge. I also would like to thank my supervisor Prof. Angelica Lim for her motivation, patience and insightful comments.

I deeply thank my caring, supportive and beloved boyfriend, Faraz, without whom this thesis would not have been possible. Most especially to my family, words alone cannot express what I owe them for their encouragement and whose love and belief in me and consistent support has been invaluable and wonderful.

It is a great pleasure for me to acknowledge and express my gratitude to AutonomyLab. My thesis has benefited from suggestions and guidances of Sepehr. I take this opportunity to thank him. I thank Pratik and Rakesh for giving me feedback on my thesis. I also have to thank Payam, Geoff, Jack for making this journey so much enjoyable.
# Table of Contents

Approval ii  
Abstract iii  
Dedication iv  
Acknowledgements v  
Table of Contents vi  
List of Tables viii  
List of Figures ix  

1 Introduction 1  
   1.1 Overview .................................................. 1  
   1.2 Hand Gestures .............................................. 1  
   1.3 Contribution ............................................... 1  

2 Related Work 3  
   2.1 Overview .................................................. 3  
   2.2 Glove-based Methods ....................................... 3  
   2.3 Vision-based Methods ...................................... 4  
      2.3.1 Detection Using Neural Networks ...................... 5  
      2.3.2 Detection Using Skeleton Tracker ..................... 6  
      2.3.3 Detection using Index Finger .......................... 7  
   2.4 Summary ................................................... 7  

3 Hand and Face Detection 9  
   3.1 Overview .................................................. 9  
   3.2 Method .................................................... 9  

4 Pointing Gesture Detection 12  
   4.1 Overview .................................................. 12
List of Tables

Table 5.1  End-to-end system error compared to ground truth: average distance on the ground plane from an actual target point to the point where the estimated pointing vector intersects the plane, at various distances from the camera.  .  .  .  24
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Pointing gesture detection for human-robot interaction</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>Example of tracking of head and hands based on skin-color classification</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>(dark pixels represent high skin-color probability) and stereoscopic range</td>
<td></td>
</tr>
<tr>
<td></td>
<td>information by Nickel et al. [23] (Copyright ©Springer 2004)</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>Skeletal tracking images extracted from Kinect for detecting pointing</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>gestures by Lai et al.[15] (Copyright ©2016 IEEE)</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>Pointing gesture recognition and target estimation by Shukla et al.[33]</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>(Copyright ©2015, IEEE)</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Hands and faces detected accurately in 15msec per frame in images from a</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>commodity UAV in flight. The user in the lower image is 10m away by MohaimanianPour et al. (Copyright ©2018, IEEE) [20]</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>System flowchart</td>
<td>12</td>
</tr>
<tr>
<td>4.2</td>
<td>Pointing gesture detection steps</td>
<td>13</td>
</tr>
<tr>
<td>4.3</td>
<td>CoBB method applied on 2D image.</td>
<td>14</td>
</tr>
<tr>
<td>4.4</td>
<td>DBSCAN clusters’ visualization in RVIZ; Blue clusters will be selected</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>and red ones are background data which will be eliminated.</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Experiment A</td>
<td>19</td>
</tr>
<tr>
<td>5.3</td>
<td>Angular Error Analysis</td>
<td>21</td>
</tr>
<tr>
<td>5.2</td>
<td>Pointing accuracy obtained with alternative keypoint selection strategies</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>(radius is the distance from camera in meters)</td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td>DBSCAN clustering results (radius in meters)</td>
<td>23</td>
</tr>
<tr>
<td>6.1</td>
<td>Robot performing pointing gesture detection</td>
<td>25</td>
</tr>
<tr>
<td>7.1</td>
<td>Distance and pointing gesture change in our evaluation</td>
<td>28</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Overview

In the field of autonomous robotic systems, human robot interaction has become an active research topic in recent years [30]. HRI is a field of robotics which addresses designing a system which involves humans and robot interaction. Human-Robot Interaction researchers and developers often seek interaction methods that are quick, intuitive and require little user training.

1.2 Hand Gestures

Hand gestures are popular due to their familiarity from everyday human-human interaction. Of these, pointing is a canonical gesture, used to direct the attention of an interaction partner to an object or place of interest [12]. Many authors have used pointing gestures in HRI systems and we survey some of these in Chapter 2. We consider the ability to quickly, reliably and accurately detect pointing gestures as an important tool in the HRI repertoire. The goal of this work is to provide a practical, reusable implementation of this component to the community, and to describe its performance.

1.3 Contribution

We describe a simple and robust pointing gesture recognition system that detects pointing gestures in individual RGB-D frames at > 30 frames per second on commodity hardware. In this project we used a state-of-the-art Deep CNN-based hands and face detector in RGB images, then by considering the corresponding depth channel pixels 3D pointing vectors are obtained. Afterwards, we test two different methods of estimating the hand end-point of the pointing vector. Then, the pointing accuracy is found by comparing with ground-truth data from a vicon motion-capture system. Finally we show an end-to-end test where a robot estimates where the person in front of it is pointing to, and report the accuracy obtained. Our source code is publically available as a ROS node, with the intention of contributing a commodity Open Source implementation of this common component of HRI systems, with state of the art robustness.
Most gesture-based HRI systems explore only close-range face-to-face interactions. We aim to provide as large a usable interaction volume as we can given the sensor capabilities. We consider the dominant version of pointing, where the vector to be communicated originates at the pointer’s eye and passes through the end of the pointing finger. A good recent empirical investigation into varieties of pointing is by Herbort [12] who states “Most participants pointed by extending the arm and putting the index finger between their eyes and the referent”.

The challenge here is that the human hand is a small, deformable object and hard to detect against cluttered backgrounds at long distances. Faces are easier to detect as they are natural fiducials, larger and less deformable than hands.

The best currently available methods for hand and face detection are based on machine learning, where feature extraction and object detection proposals are learned from labeled data in an end-to-end process. In particular, deep Convolutional Neural Networks (CNNs) have demonstrated tremendous success in object detection tasks. We employ a previously published state of the art CNN model to detect image regions corresponding to hands and faces in 2D RGB images reliably at up to 10m from the camera, where hands have just a few pixels [20]. In this thesis we consider how to use the 2D hand and face regions and corresponding depth pixels to robustly estimate the intended 3D pointing vector. We obtain a usable range of around 5m from a camera with 60 degree field of view. This usable interaction volume is large enough for both table-top and mobile robot interactions.
Chapter 2

Related Work

2.1 Overview

There have been many studies using pointing gesture detection for human-robot interaction. Previously, Pourmehr et al. designed a HRI system in which a robot looks for humans inside an area. As soon as it finds one, it approaches the person and asks for a softball. If that user is carrying the ball, he or she would proceed to offer it to the robot. If not, the robot looks for another human around the area that the user is pointing to (left or right) [28]. In this scenario, two kinds of gestures are important: one is offering the ball to the robot that can be interpreted as pointing to the robot itself, and the other gesture is pointing to someone in two opposite directions: right and left. To extend this system, we implement a pointing gesture detection algorithm in which the robot is able to detect pointing directions more accurately. Also, pointing gestures can be detected in a continuous space rather than the categorical left and right.

Zhang et al. in [41] implemented a HRI system for selecting one robot among multiple robots by gazing at them. In another work by Milligan et al. [16], selecting a group of robots from multiple robots is achieved by drawing a circle in the air around a desired subgroup of robots. In this approach robots can determine whether it was circled by the user or not. Robots are able to do that by considering the position of the user’s face; specifically, if the user’s face is located inside the shape that was drawn by the user. This work can be done by pointing at each robot in the group to form the groups. In terms of capturing human body and postures, various approaches have been proposed. We can divide the methods into two subgroups: glove-based and vision-based.

2.2 Glove-based Methods

Hand movement data acquisition is important in gesture detection. Glove system technologies are one way of acquiring hand movements easily. As glove-based methods are reliable and easy to implement, pointing gesture detection began with the help of wearable devices [29]. Researchers have continued to investigate using contact-based devices, although one of the drawbacks of using glove-based techniques is that the user needs to be trained for using the devices. Another limitation
of using glove-like systems appears to be the glove limiting the user’s haptic sense and naturalness of movements. Also, they can be heavy, and not deformable enough [7]. As an instance, one of the most accurate glove systems currently available is CyberGlove II. The CyberGlove II, developed by Stanford University is made of stretch fabric and equipped with 18 motion capture sensors on each joint of the fingers and 4 other sensors on other parts of the hand [1]. Glove-based methods are comprehensively reviewed in a survey done by Dipietro et al. [7].

2.3 Vision-based Methods

With recent achievements in computer vision, vision-based pointing detection methods are getting better. They do not need any wearable equipment, which makes interactions more human-like. Even novice users can work with it more easily and can move freely. However, recognition precision can easily be affected by occlusion, illumination and other environmental factors. Since hands are deformable objects with a considerable number of degrees of freedom (DOF), it can be challenging to handle all possible hand gestures. It also heavily depends on camera viewpoint, type of camera and camera resolution. Moreover, the trade-offs between performance-accuracy and being real-time can be hard to handle. Gesture recognition using vision-based methods are reviewed in a survey done by Mitra et al. [17], particularly hand gesture recognition is studied in [34, 30].

In vision-based methods, the choice of camera technology plays an important role: stereo camera, multiple cameras, Time-Of-Flight (TOF) camera or depth camera are different methods for pointing gesture detection.

Watanabe et al. in [38] use eight monocular cameras that are placed at an interval of 45 degrees in the horizontal plane with their optical axes crossing at the center of a square studio. By searching for skin color in the images, face and hand positions are estimated. Then by using 2 cameras directions of a pointing gesture is predicted.

Kehl et al. in [14], use multiple cameras around a portal and one camera was placed overhead. After extracting silhouettes of the user, hand and face positions are determined on the silhouettes. Then, by using the eye-fingertip line method, the direction of the pointing gesture is calculated. Multi-camera approaches are promising but less convenient for mobile-robot HRI.

A TOF camera was used in the method implemented by Droeschel et al. [8]. By detecting the face, the hand and the arm in 3D, they are able to find the pointing direction. Various methods have been used to detect pointing gestures with these sensor data, including parametric Hidden Markov Models (HMMs) proposed by Wilson et al. [39].

Cascade HMMs implemented by Park et al. [26] give good results in stereo camera data. The authors used 3D particle filters for hand tracking, then a cascade of two HMMs for finding the direction which the user is pointing to. The face and hands are tracked using particle filters, and then it passes the hand position estimate to the first HMM which maps the position to a more accurate position using kinematic characteristics of the pointing finger. Then the second HMM gets 3D coordinates
of the hands as an input, and it detects the pointing gesture. By taking into account both velocity and position information obtained from cascade HMMs, they obtained results with high accuracy. However, this accuracy is because of the large number of HMM states in the first stage, which requires relatively long processing times and a large amount of training data.

In a work done by Pateraki et al. [27], by integrating face orientation and hand gesture recognition system, two pointing gestures are detected: "point to left" and "point to right". Their face orientation system uses a face tracking approach based on the Least Square Matching (LSM) method. Skin-color regions are tracked using skin-color blob tracker. Then by using incremental Bayes classifier blobs are classified into the left hand, right hand, and face. Then hands are fed to the hand gesture recognition module, and facial blobs are fed to a Least-Squares Matching (LSM) module to find the head orientation. Pointing gesture is estimated using Dempster's rule of combination to integrate two LSM and hand gesture recognition module.

### 2.3.1 Detection Using Neural Networks

In the research done by Nickel et al. [22, 23, 24], a monocular camera is used to find the face and the hands based on the skin color, then a multi-hypothesis tracking framework finds the 3D positions of the face and the hands. Afterwards, using a pre-trained HMM classifier, the pointing detection is estimated. To improve the idea, a neural network was used to detect head pose. However, their approach uses a three feature sequence to detect pointing gesture that causes delays in detection.

More recently Richarz et al. [32] used a neural network on a monocular camera. The training phase users were asked to point to marked circles on the floor. Target points are based on a user-centered polar coordinate system: \((r, \phi)\). Valid areas for targets for \(r\) were from 1 to 3 m and a range of \([-120^\circ\] to \(+120^\circ\]) for \(\phi\). Pointing gestures were performed as a defined pose: an outstretched arm with the user indicating towards the target point. A total number of 900 images of 10 different interaction partners were labeled with distance, radius and angle. This is sensitive to pose variations and the pose that it was trained with. This method was investigated on a robot called HOROS.

In 2014, Toshev et al. [35] suggested DeepPose which uses a neural network to estimate human pose. DeepPose is based on a DNN-based regression trained with images of people. For each person in the database, their full body is considered and labelled with a total of 14 joints. This method has occasional difficulty with occlusion of the body.

Recently, Cao et al. [4] proposed a real-time multi-person 2D pose detection. Their proposed method is able to detect human body, foot, hand and face with 135 key points on single images. It detects poses in real-time, but when the body is occluded, it fails in detecting poses and can also lead to false negatives. There are other pose detection methods based on neural networks [2, 36, 21] that can be used to find pointing gestures. All these methods are sensitive to occlusion as they need to see all the body parts to determine the pose but this task can be done by using only hand and face detector. Our objective is to provide a commodifying pointing gesture detector to run on top of the all other modules that an HRI robot is operating. For this work, we have used a hand-and-face
detector based on a convolutional neural network, but it can be done using any other hands and face
detector.

![Image of skin-color map and disparity map](image.jpg)

**Figure 2.1:** Example of tracking of head and hands based on skin-color classification (dark pixels represent high skin-color probability) and stereoscopic range information by Nickel et al. [23] (Copyright ©Springer 2004)

2.3.2 Detection Using Skeleton Tracker

There are many researches that use the skeleton tracker provided by the Microsoft Kinect in order to find a pointing gesture [5, 13, 37, 40, 15]. The Microsoft Kinect skeleton tracker loses track of skeletons with occluded body parts, and also performs poorly in ranges less than 1.2 m.

In the work by Cosgun et al. [5], a pointing gesture is detected using the skeleton extractor of both Microsoft Kinect SDK and OpenNI. Their goal was to detect a correct pointing gesture when there is ambiguity with the help of two approaches: elbow-hand ray and head-hand ray. They evaluated their method with 6 users pointing in 7 directions. Also Jing et al. in [13] use the skeletal map from the Kinect sensor to detect the joints, the hands and the depth, to reduce the number of false positives in hand detection. They use a Kalman filter to detect fingertip motion. They evaluated their method by the use of a virtual screen with 6 volunteers performing pointing gestures in front of a wall with 8 lamps and a Kinect sensor.

The method proposed by Ueno et al. [37] has a training phase in which the user stands in front of the Kinect camera and points to the camera. The vertical and horizontal offsets for the individual in their work are calculated and used for further pointing gesture recognition. The skeletal data and depth map are used for this purpose.

In another paper by Wittner et al.[40] pointing gesture recognition is done using Kinect sensor and OpenNI’s NITE module. They use an Extended structure of CRF (Conditional Random Fields) called Latent-Dynamic CRF (LDCRF) with defined features like the height difference of the two hands. Evaluation was done using 18 users with range of 4m from Kinect, the overall accuracy they reported is 83%.

Indubitably, the skeletal map provided by Kinect and OpenNI is one of the more accurate ways of tracking human skeletons joints and regions of the body for gesture recognition. Although the skeletal tracker is robust from 1.2m to 3.5m, it is vulnerable to occluded body parts. In all the reviewed
2.3.3 Detection using Index Finger

Another approach that researchers have used to detect pointing gestures is detecting the index finger extending out. Shukla et al. [33] proposed a probabilistic appearance-based model trained with images captured from different viewpoints which is independent of the user’s body posture and does not require full-body or partial-body postures for detection. However, it relies on hand and finger pose, which are only available at close ranges since many hand-pixels are required. Jing et al.’s work [13], explained in the previous section, uses index finger detection to determine a pointing gesture.

2.4 Summary

Pointing gesture detection is a field of interest for many researchers. Though it has been a field of interest of researchers, there are few commodifying pointing gesture system methods that do not need special equipment. There are some approaches that use skeletal trackers but are vulnerable to
occlusion. So, it is important to implement a commodifying pointing gesture detection system that can be used off the shelf and does not need a trained user to interact with.
Chapter 3

Hand and Face Detection

3.1 Overview

People typically perform pointing gesture by extending their arm and index finger [12]. Pointing gestures can be divided into various types including bent arm and short arm pointing gestures. In this project, we considered one of the most frequent forms of pointing gestures which is the extended arm, in addition, palm and fingers are not restricted to any specific form: pointing detection could be done even with an open palm.

To detect pointing gestures in real life, only hands and arms are needed, and people do not need to see all parts of the human body to understand to which direction the person is pointing. In this thesis, detection is done using only the hands and face of the person who is pointing, so that occlusion of other body parts will not be a problem. Our purpose is to detect pointing gestures in a fast and intuitive way.

3.2 Method

In our computations for detecting pointing gestures, we require one pointing hand and the face of the user, which allows us to detect pointing gestures even when other body parts are occluded.

We employ a robust state-of-the-art hands-and-face detector implemented and proposed in S. MohaimenianPour’s MSc thesis [19]. This section describes his work in summary which is the basis of the pointing gesture detection. This face-and-hand detector is based on a deep Convolutional Neural Network (CNN) derived from YOLOv2. [31] YOLOv2 is an object detection system which generates object detection candidate bounding boxes, with the probability of the presence of each object label.

This network was modified and re-trained by hand-labeled data for face and hand detection that is so reliable and robust that stable hand and face detections can be obtained in almost all frames [20]. Detection is done in every frame, with about 15 msec computation time on a commodity GPU
False positives are very rare. For increased robustness and smoothing, a bank of Kalman filters was applied that tracks detections from frame to frame at 30Hz. The output of this system is a set of Regions of Interests (ROIs) corresponding to each detected hand and face in a 2D image. When a person is in the field of view of the camera, their face and hands are very reliably located in the RGB image.

The CNN network implemented and proposed by MohaimenianPour et al. was trained on the largest integrated hands-and-faces detection dataset with labeled drawn bounding boxes as ground-truth provided by S. MohaimanianPour. The data were gathered from different sources. A hand-labeled dataset specific for human-UAV interaction was created and annotated, and in addition other datasets have been annotated and used for training, including frames with a vast variety in distance, illumination, background, pose, resolution. Some of the sources are Mittel [18], Pascal VOC [10] and Faces in the Wild [3]. Approximately, the dataset consists of 50,000 frames which are annotated and labeled. Hands and faces which are at least 50% visible are called detectable and annotated with tight bounding boxes.

After gathering data, modifications were done on YOLOv2 so that detecting both hands and face
was possible. As a result, hands and face were detected using a monocular camera in 60fps for image size of 736 x 736 pixels that makes hand and face detector applicable for real-time use in robotics. As shown in Fig. 3.1, this system can detect hands and faces up to 10m. The hand and face detector is available as http://autonomylab.org/hands_and_faces.
Chapter 4

Pointing Gesture Detection

4.1 Overview

We seek to detect pointing gestures by considering only face and hands. After detecting the user’s hands and face in the 2D image, we consider corresponding locations of the bounding boxes in the depth image derived from a 3D sensor. It is applicable since the RGB and depth images are pixel-aligned. When two hands are present in a frame, we choose one and assume it is performing the pointing gesture. By observing pointing gestures in the real world, we concluded that in most cases the hand which is performing pointing gestures is in the higher position than the other hand. Therefore, as a simple heuristic, we use the hand closest to the top of the image as is natural in most cases. More interesting heuristics are easily substituted. Given pixel locations and depth, and cali-

Figure 4.1: System flowchart
brated camera intrinsics, we obtain point clouds for each region that expectedly contain points on either the hand or face. Consequently, as we only obtain point clouds inside the ROIs, we save large amounts of time to process data and points. Since hands are small, we typically have only a few sparse points for the pointing-hand. Then, we estimate the intended pointing vector in 3D starting from some point in the vicinity of the head and passing through some point in the vicinity of the hand. These two keypoints must be selected, but it is not obvious how to decide them. To find out how to select points, we tested different approaches that will be explained further. This method is similar to the 2D method using the eye-finger line described in the paper by Couture et al. [6], but here in 3D.

The hands and face bounding boxes often contain depth points that are either from the background behind the user, and spurious depth estimates from an imperfect sensor or even points from objects that occlude a portion of hands or face. Since we are processing the points inside the bound-
ing boxes and our assumption is that points represent the depth of the hands and face, we need to eliminate these irrelevant points. We address this problem with two simple but effective complementary approaches: 1) considering only points that lie close to the center of the bounding box in a 2D image (CoBB); and 2) clustering points in depth using DBSCAN (Density-based spatial clustering of applications with noise) [9] to eliminate outliers. Fig. 4.2a and a system flowchart shown in Fig. 4.1 illustrate the overall approach. In the following section, we describe these methods in detail.

4.2 Considering the Center of the Bounding Boxes (CoBB)

The first approach is a simple geometrical heuristic. It is naive but fast. The output of the CNN model usually has hands and face located in the center of their detected bounding boxes. Bounding boxes are tight axis-aligned rectangles around the visible part of each object of interest in the image, and that the bounding boxes are slightly larger than the objects they contain. Thus pixels far from the center of the bounding boxes and mostly on edges of the rectangle are more likely to be background or spurious points, and points close to the center are more likely to be correct hand or face target hits representing more precise depth data. To filter out the unwanted points we consider a circle with radius $r$ in the center of bounding boxes, and ignore the points outside of the circle as shown in Fig. 4.3. Parameter $r$ was hand-tuned by experiment: it has to be large enough so that it could contain some points at large distances where depth data is very sparse, but small enough to eliminate background pixels with a high probability.
Through some experiments the radius of circle $r$ was chosen. First, $r$ was considered as a percentage of the bounding boxes width. But since hands can be in a position where the width of the rectangle is larger than the height, we estimated the radius based on width or height whichever is smaller. Tested values of $r$ include 70%, 50%, 35% and 10% of the width/height. Among these, $r$ larger than 35% of the smaller of the width and height of the rectangle could not help in eliminating unwanted points since data points of background still exist in the rectangle. Also, smaller values cover a few points as the user stands further from camera and in some cases no data point falls in the circle. We found that $r = 35\%$ of the smaller of the width and height of the rectangle worked well, so we take into account points inside the circle and ignored points out of that range. This simple and fast (O(n)) technique serves to filter out most background and spurious points.

Having removed most outliers, we find the two keypoints that define the pointing vector based on the remaining pointclouds. The angle to the keypoint is obtained by the geometric mean in pixel-space $(p, q)$ and depth of the clustered points, projected using the camera intrinsics. To find the depth we compared three straightforward approaches: 1) mean, 2) median and 3) minimum depth to the camera in each of the face and hand regions (closest point to the camera). The experiment and results are described in Chapter 5.

4.3 DBSCAN

The second approach uses explicit clustering in an effort to improve performance with relatively poor data at large distances from the camera. As explained earlier, the camera’s specification states good quality depth perception up to around 2.8m. At greater distances the error increases rapidly, and very few accurate points are returned. So we struggle to find the outliers in larger ranges. As explained above, background data points are unwanted points inside each bounding box. Additionally, spurious points from our sensor are another set of points which we want to discard. Our previous approach can lead us to find points that are mostly in the center and not in the edge of bounding
boxes, but to eliminate garbage points produced by sensor we need to cancel out outliers which can be even in the assumed circle inside bounding boxes. To achieve our goal, we consider clustering points in each bounding box. By implementing K-means clustering [11] on points acquired from our depth images, we were not capable of finding our desired clusters that could properly represent depth of hands and face. The K-Means clustering algorithm partitions points inside ROI into $k$ clusters with two objectives of making each cluster as compact as possible and the clusters as separated as possible. Eventually, K-means underperformed in this task, since a lower $k$ did not exclude outliers from clusters, and with a higher $k$ discriminating between clusters to find main clusters turned out to be problematic since the main cluster would merge with outliers to have at least $k$ members.

In our next attempt, a clustering algorithm named DBSCAN was implemented. DBSCAN, which stands for Density Based Spatial Clustering of Applications with Noise, is a popular robust generic clustering technique [9]. In this algorithm, a cluster is defined by the density and connectivity of data points; it bundles points in high density regions together and any point that lies alone in low-density regions and is not part of a cluster is rejected as noise. The algorithm requires two hyper-parameters to be set; one is minimum number of points inside a cluster and second one is $\varepsilon$ value. Basically, points in distance $\varepsilon$ neighborhood of each point group as a cluster, and if any cluster does not satisfy the specified minimum number of points, it will be discarded.

As DBSCAN does not require a number of clusters, it is an applicable algorithm for our objective. We apply it to find clusters of points that correspond to the target, and reject background points and also sparse spurious points derived from camera. DBSCAN clustering can find arbitrary numbers of arbitrary shaped clusters, each with a specified minimum number of points, and reject outliers, so that if we use DBSCAN to cluster points according to their depth, we can reliably obtain a foreground/target clusters as hand/face. Background and objects that occluded them, and all spurious points will be discarded. As mentioned before, we cluster points based on their depth and distance from camera. We assume the cluster with highest number of points to be the hands and face, since bounding boxes are tight rectangles around hands and face. Alternatively, depending on the environment, we could take the closest cluster to the camera assuming background is another big cluster that we obtain from our algorithm.

Finally, now that we have set of points indicating depth of hand and face, the keypoint is computed as the geometric mean in pixel-space $(p, q)$ and depth of the clustered points.
Chapter 5

Evaluation

5.1 Overview

To evaluate our approach, we performed two experiments. The first experiment (Experiment A) evaluated how well we estimate the hand and face positions, agnostic to the user’s intended pointing target. The second experiment (Experiment B) was an end-to-end test to validate the applicability of the approach for HRI applications.

5.1.1 Experiment A

This experiment compared the output of our system to a motion-capture system ground truth. Specifically, we measured a lower bound on the pointing vector accuracy by comparing the vector obtained from our vision system with the vector obtained from a vicon motion capture system\(^1\). Two participants (with heights 186cm and 170cm) stood in 25 pre-set positions relative to the camera, covering its practical field of view which is around 60 degrees in its horizontal direction as shown in Fig. 5.1a. The closest position to the camera was less than 1 meter so the users’ body was occluded but face and hands were partially visible. Other positions were farther away and the farthest ones were 5.5 meters from the camera. Users stood in each position with a vicon-marked helmet and vicon-marked rigid object in their hand. The user pointed for two seconds to each of four different directions as shown in Fig. 5.1b.

5.1.2 Experiment B

In this experiment we evaluated how well our system worked in a real world HRI scenario. The users were asked to point to three fixed targets marked on the floor of the experiment area. A robot calculated the intersection of the detected pointing vector with the ground plane, to determine where the user was pointing. We report the error distance between the intended point on the floor and the robot’s estimate. The experiment was repeated for the same users from experiment A.

\(^1\)https://www.vicon.com/
5.2 Setting

In these experiments, an Intel Realsense ZR300 camera was used as an RGB-D sensor and it was mounted on a pioneer robot. All experiments were inside a robotics lab in front of a bookshelf which produced a very cluttered background.

For the first experiment, we used the vicon Motion capture System. The vicon system is designed to track movements. The system contains a set of cameras and markers, and vicon cameras were installed around the lab. Markers are spheres covered with silver reflective paint and are attached to the objects in the field of view of cameras. Cameras track and reconstruct markers visible in their field of view in a 3D space (on the computer connected to the cameras). The visible markers appear highly illuminated in the virtual 3D space in the computer screen. After capturing the coordinates of each marker vicon system links the chosen markers and forms an arbitrary shape, then the object is trackable and conspicuous in the 3D space. From then on, the coordinates of each object is reported from the cameras.

In our experiments we used a tight-fitting helmet and a pair of scissors each covered with vicon markers to track users’ position and pose. The vicon tracking configuration is shown in Fig. 5.1c.

5.3 Experiment A: Pointing Angle Accuracy Compared to vicon

In our first trial, we evaluated our system using the CoBB method (Section 4.2). After finding the bounding boxes around hands and face as explained in Chapter 4, we used CoBB method to ignore outliers. To retrieve the positions of hands and face using keypoints, we compared the results using the a) mean, b) median of the points or c) simply picking the point closest to the camera. Two participants were asked to stand in the 25 positions and point to the 4 different directions, including pointing away from the camera and to the back until only a part of the hand was visible to the camera, as shown in Fig. 5.1b.
(a) Setting of 25 positions with respect to the camera.

(b) The four pointing directions performed in Experiment A. On the right, the approximate pitch of pointing gesture is shown. On the left, we depicted the approximate yaw.

(c) The user wore a helmet and carried a pointer, both instrumented with vicon reflective markers

Figure 5.1: Experiment A
We tested our system up to 5.5 meters from the camera and ± 30 degrees as the horizontal field of view of the Intel Realsense ZR300 camera is 68 degrees. The sample points are shown in Fig. 5.1a. The results of this experiment were gathered and analyzed by comparing our results (pitch and yaw of hands with respect to the face), with the ground truth (pitch and yaw calculated from vicon markers). Each pointing gesture direction can be represented using the yaw and pitch of a vector passing from eyes to pointing hand. The roll of the vector is not important in pointing gesture detection. The error is reported as the mean of absolute differences in the angles. To accomplish this, as positions of hands and face in the vicon space are different from the global coordinate frame, vectors obtained from the helmet and the rigid object should be transformed to the coordinate system of the eye-to-hand vector. For the purposes of this, the rotation matrix R is used (rotation by angle A degree about the axis x, a rotation by angle B degree about the axis y, and a rotation by angle C degree about the axis z):

\[
R = \begin{bmatrix}
\cos C \cos B & -\sin C \cos A + \cos C \sin B \sin A & \sin C \sin A + \cos C \sin B \cos A \\
\sin C \cos B & \cos C \cos A + \sin C \sin B \sin A & -\cos C \sin A + \sin C \sin B \cos A \\
-\sin B & \cos B \sin A & \cos B \cos A
\end{bmatrix}
\] (5.1)

After transforming the vector derived from the vicon to the same coordinate frame of the pointing vector derived from our system, we find the measured angular error varied from 0 to 10 degrees. The results are shown in Fig. 5.2 and Fig. 5.3. Although results are not reported based on the exact position, we assume there is no significant difference in results in the same distance at a slightly different angle. Positions in the middle column were exactly in the middle of each region and the side columns were close to the very end of each side.

The results of using CoBB method explained in Section 4.2 using three explained approaches of finding mean, median, or finding minimum depth to the camera, are shown in the Figures 5.2, yellow indicates highest error, which is a difference of more than 20 degrees which is not acceptable; darker blue indicates a more precise estimated pointing vector. The plots depict that the most precise results are in distance of 2 to 3 meters away from camera, which is due to reliable points obtained from depth data in that range. Also, the left side of all plots seem to have larger error. Based on our observations is more of the vicon system’s flaw in detecting the markers in the side of the lab which is close to the boundaries of field of view of vicon cameras and also cluttered. Therefore, on the right side of the room the results are more precise. We could achieve better results in the middle column in all three phases of experiments. Apparently, in these three strategies using CoBB, finding mean depth showed the best result. Also remarkable output was achieved in distances more than 5 meters, in finding the keypoints using median of the points. Using the closest point to the camera in distances more than 5 meter our method underperformed since in larger distances only a few number of data points exists, although better results achieved using mean depth. The results
show that calculating keypoints using the mean depth of inlier points was more accurate, especially farther from the camera.

Results of the DBSCAN clustering approach are reported in Fig. 5.4. Similar to other experiments, it was performed with two people in 25 positions and performing four pointing gestures. Observing previous results, we use mean operation on the points inside the chosen cluster. The results show that DBSCAN clustering improves accuracy compared to the geometric CoBB approach. However, at larger distances, the clustering method fails to detect a cluster in many video frames, since the sensor does not provide enough points and as described before (Section 4.3) for running the DBSCAN algorithm on a set of points a minimum number of points is required, so in distances, more than 4.5 meters points are too few. At distances below 4m, we are able to obtain clusters and thus pointing vectors for almost every frame at 30Hz. At distances of 4m to 4.5m, due to depth data being only occasionally available, we obtain pointing vectors at around 10Hz, hence the users were asked to stay in pointing gesture for almost 10 more seconds. According to the camera specifications depth data is accurate up-to only 2.8m. Apparently, as shown in the Fig. 5.2 DBSCAN algorithm performed noticeably more accurately than other geometric method.

5.4 Experiment B: Pointing to the Targets on the Floor

Since one of the common HRI applications of pointing gesture detection is for commanding a robot to reach to a goal point, we validate our system using marked points on the ground with a known position with respect to the camera.

To estimate goal points the intersection of the 3D vector of pointing gesture with the ground is calculated. This is achieved by intersecting the pointing vector with the ground plane at $z = 0$: 

![Figure 5.3: Angular Error Analysis](image-url)
Figure 5.2: Pointing accuracy obtained with alternative keypoint selection strategies (radius is the distance from camera in meters)
**Figure 5.4: DBSCAN clustering results (radius in meters)**

\[ Z'_{\text{face}} = Z_{\text{camera}} + Z_{\text{face}}, \]
\[ Z'_{\text{hand}} = Z_{\text{camera}} + Z_{\text{hand}} \]

(5.2)

Where \( Z_{\text{face}} \) and \( Z_{\text{hand}} \) are the heights of the face and the hand obtained from our system, and \( Z_{\text{camera}} \) is the height of the camera from the ground which in our experiment is 1.2m. The result is the actual height of the face and the hand (\( Z'_{\text{face}} \) and \( Z'_{\text{hand}} \)) from the ground.

\[ \vec{P} = (\text{face} - \text{hand}) \]

(5.3)

Where \( \vec{P} \) is the line passing through hand and face.

\[ t = \frac{Z'_{\text{face}}}{P_z} \]

(5.4)

Where \( t \) is the scale ratio used in equation below to calculate the relative goal point(intersection point).

\[ G = (X_{\text{face}} - (t \cdot P_x), Y_{\text{face}} - (t \cdot P_y), 0) \]

(5.5)

As ground truth, the target points on the floor were measured by hand and marked with tape. The
target points were placed in different positions, including on the ground behind of the robot. In this experiment, two people were asked to stand in 5 different positions, pointing to 3 marked points on the ground. The goal point detected by our system was compared with known positions using Euclidean distances between goal points and marked points. Results are shown in Table 5.1. The variation of error detection was between 0.008 to 0.01, which shows the robustness of the system. These results are comparable in accuracy at close ranges to other reported systems listed in [33]. However, our system has a larger usable envelope and runs at the frame rate of the RGB-D sensor.

Table 5.1: End-to-end system error compared to ground truth: average distance on the ground plane from an actual target point to the point where the estimated pointing vector intersects the plane, at various distances from the camera.

<table>
<thead>
<tr>
<th>Distance (m)</th>
<th>µ (cm)</th>
<th>σ (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5</td>
<td>16.1</td>
<td>1.9</td>
</tr>
<tr>
<td>2.5</td>
<td>18.1</td>
<td>2.1</td>
</tr>
<tr>
<td>3.5</td>
<td>14.5</td>
<td>3.5</td>
</tr>
<tr>
<td>4.5</td>
<td>22.4</td>
<td>5.6</td>
</tr>
<tr>
<td>5.5</td>
<td>48.4</td>
<td>12.3</td>
</tr>
</tbody>
</table>
Chapter 6

Demonstration

Having detected a target point on the floor, we complete our end-to-end robot system by having a robot navigate to the target. The robot measures the covariance in the pointing direction estimate for the most recent 30 frames (1 second). If the robot can collect a batch of 30 frames in 1 second in which the covariances fall below a threshold, the robot navigates to the indicated goal point. After reaching the goal point it will then start observing around to find another human to find a pointing gesture. For the work described here, we use a Pioneer DX3 robot carrying a laptop on top as shown in Fig. 1.1 with the Realsense ZR300 RGB-D camera connected to the laptop with a commodity GPU (NVIDIA GeForce GTX 1060) which provides the computational power needed for running the deep neural network for hands and face detection. The mobile robot base is controlled by the built-in computer running ROS (Robot Operating System).
Chapter 7

Conclusion

7.1 Overview

The goal of this work was to provide a practical, robust approach to pointing for HRI applications. We demonstrated a system and evaluated its performance.

We presented a method for pointing gesture recognition in a complex environment with cluttered visual backgrounds and varied lighting, at ranges up to 5.5m. We compared two approaches for outlier rejection and determining the key points of the pointing vector. A clustering approach using DBSCAN was found to be more accurate but gave results in only a subset of frames when the user was far from the camera. A geometric heuristic gave results at a higher rate but with a less accuracy at larger distances.

7.2 Future Work

Our systems were shown to work with good accuracy, although there is room to improve the robustness considerably. Hyper-parameters that should be specified in the DBSCAN implementation (minimum numbers of points and \( \varepsilon \)) could have a high influence on the final result, so it would be useful if one could modify those parameters during running time depending on how far a user is standing. In cases when a user is close to the camera, clusters are bigger and minimum number of points required to be inside of a cluster can be greater. We would obtain similar benefits from using the same method for finding \( r \) of the CoBB method.

While our pointing gesture detection is fast and robust, in practice we developed a single-task robot which always seeks for pointing gestures. Our work can be extended to use cues in order to understand if the human actually intended to point to somewhere, for instance, using natural language processing.

As our general purpose is to make robots that are smarter, more human-like and even safe to interact with, our system can be a basis for several applications in human-robot-interaction. In [25] a system is implemented so that robot follows not behind the humans but ahead of them and in some scenarios robot loses track of its user and tries to correct itself by its constructed map of area
and by taking into account other possible ways user has chosen. This would be extended in a more intuitive approach, in which a user points to the direction it aims to go and the robot follows the command, with lower chance of losing track of human it is following. Also, it can be applicable in other scenarios like finding a target point, as it was one of the experiments. This system could be extended with an object recognition system so that robot will search for a specific object in the area the user is pointing. This can help the elderly pick up objects when they point to them. There are many other applications that pointing gesture can be applied to.

### 7.3 Code and Reproducibility

This system is offered as a commodity component for HRI systems, with state of the art speed and robustness, and comparable performance to other systems at close range, but a larger usable interaction envelope. Source code including a ROS node is provided at [https://github.com/AutonomyLab/pointing_gesture](https://github.com/AutonomyLab/pointing_gesture). The commit hash for the version used to obtain the results in this paper is 7a4fe3a102c528c606cb3cac6e91cedc8d54b80a.
Figure 7.1: Distance and pointing gesture change in our evaluation
Bibliography


