Hand gesture commands detection for Robotic applications

Ekaterina Kapanzha, Luz Maria Martinez Ramirez, Bare Luka Zagar

Abstract Nowadays robots and robotic applications are more and more common in our everyday life. The communication with the robot has to be more intuitive and user-friendly than the common communication methods like keyboard and mouse. Recently the popularity of the voice commands and hand gestures as human-machine communication interfaces gain more and more popularity.

This work is concentrated on hand gestures detection. We propose an algorithm that combines the machine learning techniques for finding the hand on the input image from the camera and 21 keypoints corresponding to the joints of the fingers and some geometrical properties and relationships between these joints to detect whether the finger is closed or open. Then using the binary sequences for different combinations of open and closed gestures are used to find the corresponding gesture.

Keywords gesture detection, robotics, machine learning

1 Introduction

The terms gesture and gesture recognition are often found when a person interacts with a computer. Gestures are a movement of the body or a form of a user's physical action to transmit some information. In the case of robotics gestures, for example, can be used to give commands to a robot.

Gesture recognition is the process by which the system reads and understands a gesture performed by the user. Through the use of computer vision, hand gestures, as a replacement for the standard input methods or as addition to them, have attracted much attention in various applications. The standard tools and methods of user-machine interaction most commonly include mouse, joystick, keyboard and electronic pen. But with the development and realization of the virtual environment and augmented reality applications these tools are not enough anymore and should include more intuitive means of human-machine interaction, for example, gestures or voice commands. In the same time, a gesture command can be more precise while representing more complicated command than just one word. In such situations the hand gesture can be more advantageous than voice command.

Combination of the different forms of fingers, palms and hands in general, identified by the gesture recognition system and then interpreted by computer or robot, has the great potential to provide a more natural human-machine interface, because for us, humans, hand gesture is a natural way to represents ideas and actions very easily. If for some time we ignore the world of computers and consider the interaction between people, we can simply understand that we use a wide range of gestures in everyday personal communication.

The gestures vary widely between cultures, and the context is still used in communication. The meaningful use of gestures in our daily life as a form of interaction motivates the use of the gesture recognition interface in robotics and computer systems and applies it to a variety of applications through computer vision.

Gesture recognition is usually performed by use of mainly three methods: (i) glove-based wearable devices [1], (ii) 3-dimensional locations of hand keypoints [8] and (iii) raw visual data. Method (i) has shown to provide good results in terms of both accuracy and speed, but it requires to wear an additional device with lots

Ekaterina Kapanzha
Technical University of Munich
E-mail: ekaterina.kapanzha@tum.de

of cables. Method (ii) requires an extra step of hand-keypoints extraction, leading to additional time and computational cost. Lastly, for (iii), only an image capturing sensor is required, which is most commonly a camera, infrared sensor or depth sensor, and those sensors are independent of the user. Considering the absence of the need to wear any additional device for tracking, this option seems to be the most practical one. And for any gesture recognition system practicality is often the crucial performance measure.

In this work we have developed a vision based gesture recognition application using RGB-camera and machine learning techniques. As the source of hand tracking in real-time dynamic environment we use pretrained model from Google [3] which provides 21 3D-keypoints correlated to finger and palm joints and then we develop the library of gestures for the command giving to a robot.

The key contributions of this paper are:

1. Although a pretrained model MediaPipe [3] for a hand tracking is available for mobile (namely iOS and Android OS) and Linux Ubuntu desktop platform, it does not have specific ROS2 implementation. Among other tasks this work also concentrates on the creating a ROS2 wrapper for the MediaPipe hand tracking model.
2. Processing of the recognised hand object, namely determining whether the finger is open or closed and respective combinations of the fingers to establish the potential gesture.
3. Creating a library with the most commonly used gestures for the giving commands to a robot, such "stop", "go", "left", "right", etc.
4. Transforming an identified gesture into a standard set of ROS2 commands for a robot.
5. Wrap up the whole system into ROS2 environment using model-based approach.

This paper is organized as follows: Section 2 presents related work. Section 3 describes our approach while Section 4, the experiments and results. Finally, discussion and conclusions are given in Section 5.

2 Related Work

Existing algorithms of gesture recognition and extraction from raw visual data can be roughly classified into two big categories: using shape analysis techniques (i) and deep learning techniques (ii). This work aims to combine the advantages of both approaches.

Two examples of the first method are Panwar et al. [5] and Vishwakarma et al. [7]. Both papers are using the shape parameters of the hand to extract a gesture from the image, while Vishwakarma et al. [7] additionally uses texture evidence, specifically the skin color of the representatives of different nationalities. The idea of encoding the hand shape, described in Panwar et al. [5] is close to the one implemented in this paper, although instead of using the relative length of each finger for the encoding of gesture, we are using the distances between the different joints of different hands (between palm and fingers and different joints of each finger).

Examples of the second approach are Xu et al. [9], Molchanov, Yang, Gupta et al. [4], Köpüklü, Gunduz, Kose et al. [2] and Bazarevsky and Zhang et al. [6]. Approaches [9] and [6] are concentrated on extracting gestures from each image of the camera, and the approaches [4], [2] are using the whole video clip to extract spatial-temporal features, which means they use not only the the object recognition in each frame (spatial), but also the changes in gesture from frame to frame (temporal), which allows those approaches to extract also complex gestures involving the position change of the hand, for example, zooming in and out gesture. These algorithms use the video sequence as a input and then perform classification using convolutional neural networks. While the results of these algorithms are more advanced, their implementations require more expensive hardware. For example, [4] uses 4 sensor input flows simultaneously: a SoftKinetic depth camera (DS325) recording depth and RGB frames, and a DUO 3D camera capturing stereo IR. And for the processing of the input signals NVIDIA DIGITS DevBox with four Titan X GPUs, a so-called "deep learning machine" is used.

As well as in the approach [6], in this paper we are using open source library MediaPipe [3] from Google for hand tracking, specifically 21 3D-keypoints, namely joints between the palm of the hand and fingers and joints of each finger.

3 Methodology

The System overview can be seen from the fig.1. The system consist of 4 modules: three ROS2 nodes (HandTrackerComponent, GestureDetectorComponent and RVIZVisualizationComponent) and one library (MediapipeLibrary). Communication between nodes is implemented using two strategies: ROS2 network with messages (marked with green in the fig. 1) and inter-process communication (IPC) via socket (marked yellow).

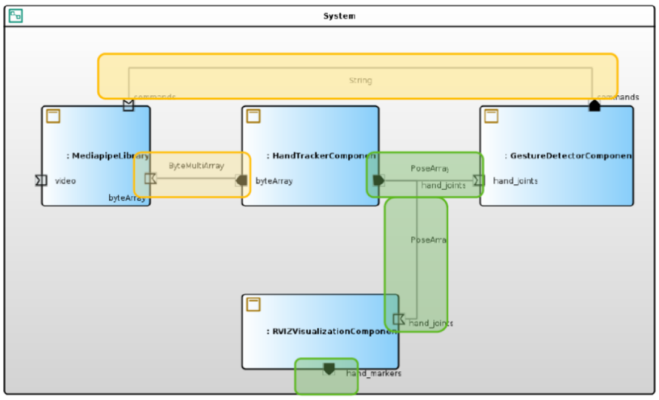


Fig. 1: System includes three ROS2 nodes and one library. Yellow indicates socket IPC and green ROS2 messages.

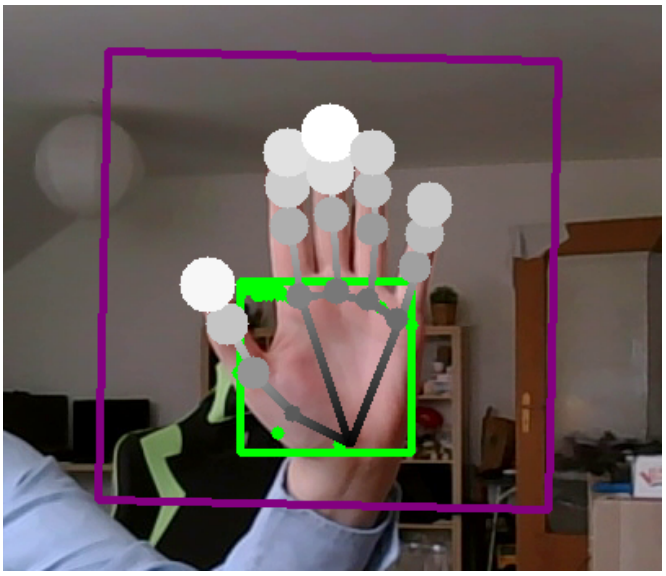


Fig. 2: Bounding boxes and produced 21 points.

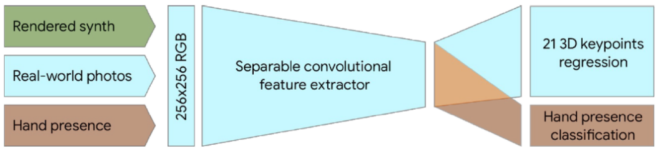


Fig. 3: Architecture of ML pipeline.

3.1 MediapipeLibrary

Mediapipe library [3] – is an open source library, which provides machine learning solutions for detection and tracking. For our project we are using single and multi-hand tracking modules of Mediapipe.

With MediaPipe perception pipeline can be built as a directed graph of modular components, called Calculators. Mediapipe comes with an extendable set of Calculators to solve tasks like model inference, media processing algorithms, and data transformations across a wide variety of devices and platforms.

Mediapipe hand tracking solution utilizes an ML pipeline consisting of two models working together: A palm detector that operates on a full input image and locates palms via an oriented hand bounding box. A hand landmark model that operates on the cropped hand bounding box provided by the palm detector and returns high-fidelity 2.5D landmarks (fig. 3).

Fig. 2 shows the resulting bounding boxes and detected joints.

3.2 Node HandTrackingComponent

Node HandTrackingComponent works as a wrapper for a Mediapipe library. Mediapipe library is modified to write the 63 values to the IPC socket (21 detected points, 3 coordinates each). Node HandTrackingComponent retrieves those points and publishes them into ROS2 network as PoseArray, topic /hand_joints. Two sockets were created for sending and receiving information in the library.

For receiving information from the Mediapipe library, a server was configured as a stream of data on port 50000 inside the handTracker_wrapper node.

On the Mediapipe library side, the implementation of the communication function receives 63 floating point values (21 points representing the hand, each point composed of x, y and z coordinates). It then converts the floating point values into a byte array for sending.

The node receives a byte array containing the defections and converts it to PoseArray messages and publishes them into ROS2 network. For every successful detection, a stream of bytes is created, a socket is opened, and the information is sent to the ROS2 node for recognition.

For receiving the results of the gesture recognition process, another server was configured as a stream of data on port 40000 inside the Mediapipe library. This function receives a byte array coming from the node GestureRecognitionComponent containing a final detection. The byte array is converted to a string value before it can be merged into the image frame for display. It then displayed on the upper left corner of the image.

3.3 Node GestureDetectionComponent

Node GestureDetectionComponent subscribes to /hands_joints and performs gestures detection. It also

writes detected gesture into another IPC socket. Then Mediapipe library accesses the socket to read the string with the name of detected gesture. This name is then inserted into the upper left corner of the image.

The gesture detection is performed via combinations of different states (open or closed) for each finger. Upon receiving the coordinates of 21 keypoints the algorithm identifies the state for each finger in the image frame.

Let us introduce some variables for identifying each of the points. The coordinates of joints of each finger can be represented as Tip - tip of the finger, followed by $MidTip$, then $MidBase$ and $Base$ being on the place of connecting the finger to the palm of the hand. Then the upper index of these variables will represent corresponding finger, for example, Tip^{thumb}, $MidTip^{thumb}$, $MidBase^{thumb}$, $Base^{thumb}$. The fingers are respectively called thumb, index, middle, ring and little. And finally the lower index represents x, y or z coordinate of each point, for example, $MidTip_x^{thumb}$. Additionally, the lowest point on the base of the hand is defined as $Palm$.

Then the procedure of detecting the gesture is as follows:

1. The width of the hand is calculated to identify the threshold for further comparisons by the Euclidean (equation 1) distance between the middle-base joints of the index and little finger, as it shown in fig. 4a.

$$width = ||MidBase^{index} - MidBase^{little}|| \quad (1)$$

2. Then for fingers index, middle, ring and little we find the distance between the tip of the finger and the palm point (fig. 4b), for example, for index finger it will be:

$$distance = ||Tip^{middle} - Palm|| \quad (2)$$

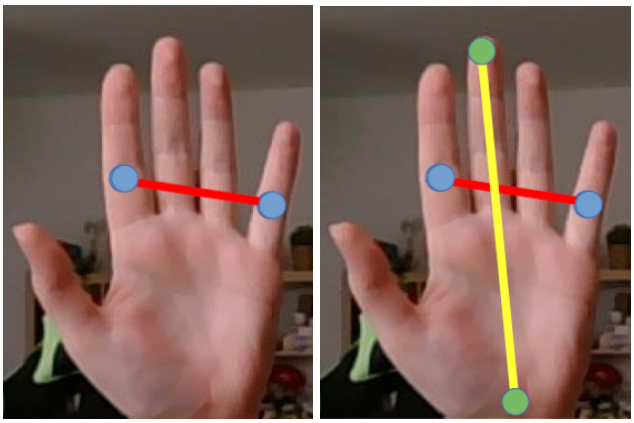
3. Two values are compared. In case the width of the hand is bigger than the distance between the tip of the finger and the palm, then the finger is considered to be closed. Otherwise the finger is considered to be open.
4. The experements showed that the same algorithm does not work for the thumb, because sometimes depending on the position of the hand on the image, width appear to be be bigger even when the thumb is open. So, first we define two distances:

$$distance1 = ||Tip^{thumb} - MidBase^{middle}|| \quad (3)$$

$$distance2 = ||Tip^{thumb} - MidBase^{thumb}|| \quad (4)$$

Then, for thumb consider to be open the following inequality has to satisfy:

$$distance1 > distance2 \quad (5)$$



(a) Width of the hand is used as a threshold (b) Detecting whether the finger is open

Fig. 4: Different distances used in calculations

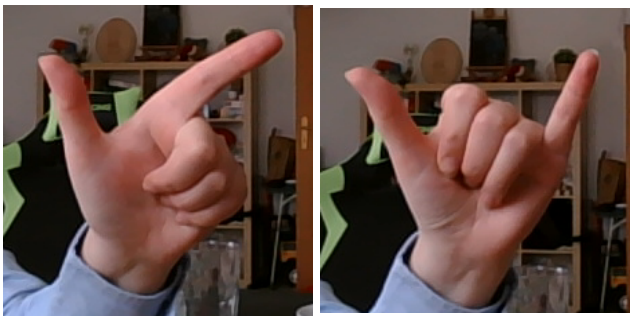
5. After detecting for each finger either it is open or closed we can further build a bit array, indicating 1 or 0 for open or closed for each finger.
   - 11111 - all fingers are open, this combination got assigned gesture "Stop" (fig. 5a)
   - 10000 - thumb is open, other fingers are closed: this combination got assigned gesture "Ok, go!" (fig. 5b)

(a) "Stop" (b) "Ok, go"

Fig. 5: Gestures used in the project

   - 11000 - thumb and index are open, other fingers are closed: this combination got assigned gesture "Scroll" (fig. 6a)

– 10001 - thumb and little are open, other fingers are closed: this combination got assigned gesture "Phone" (fig. 6b)



(a) "Scroll" (b) "Phone"

Fig. 6: Gestures used in the project

– 01000 - index is open, other fingers are closed: this combination got 3 gestures assigned. Default gesture is "Attention" (fig. 7). Additionally the x-coordinate of the tip of the index finger was checked. In case if its x-coordinate was the smallest, hence, the tip was the furthest left, then the assigned gesture is "Left" (fig. 8a). Oppositely, if its x-coordinate was the biggest, hence, the tip was the furthest right, then the assigned gesture is "Right" (fig. 8b).

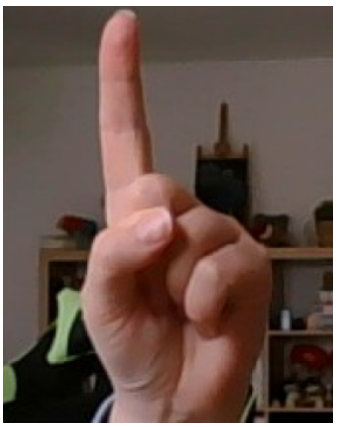
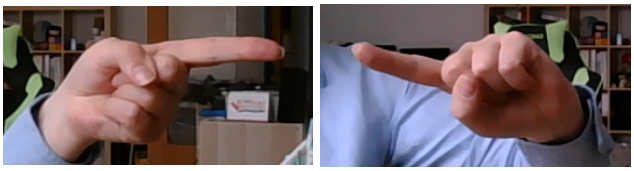


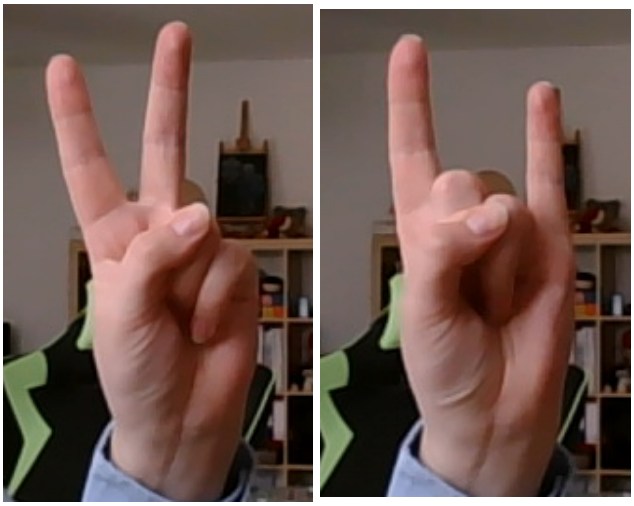
Fig. 7: "Attention"

– 01100 - index and middle are open, other fingers are closed: this combination got assigned gesture "Peace" (fig. 9a)
– 01001 - index and little are open, other fingers are closed: this combination got assigned gesture "Rock" (fig. 9b)



(a) "Left" (b) "Right"

Fig. 8: Gestures used in the project



(a) "Peace" (b) "Rock"

Fig. 9: Gestures used in the project

3.4 Node RVIZVisualizationComponent

Node RVIZVisualizationComponent converts geometry_msgs/PoseArray into visualization_msgs/MarkerArray for better visualization in RVIZ, where the keypoints of each finger get assigned with specific color (fig. 10).

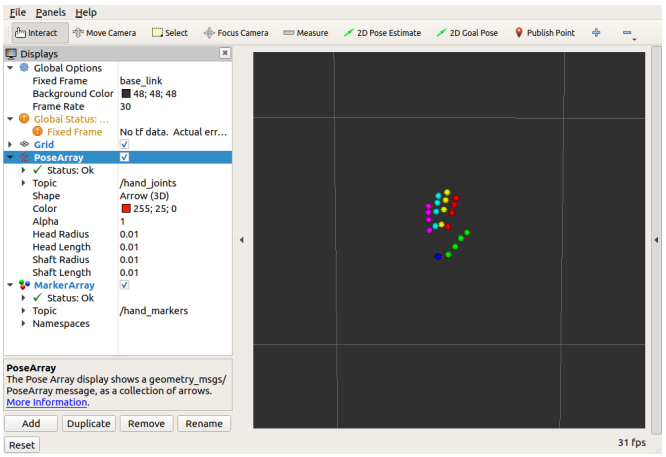


Fig. 10

4 Experiments

The gesture detection algorithm was tested in different scenarios with different levels of complexity, for which a few videos were recorded as well as testing of the algorithm in real-time using a stream from the web-camera was performed. The testing videos can be categorized as follows:

1. The best performance was shown in the good lightning conditions without other objects on the background (white wall), where algorithm detected 90 out of 100 gestures showed one after another in the random sequence, which corresponds to 90% of the correct detection (fig. 11).

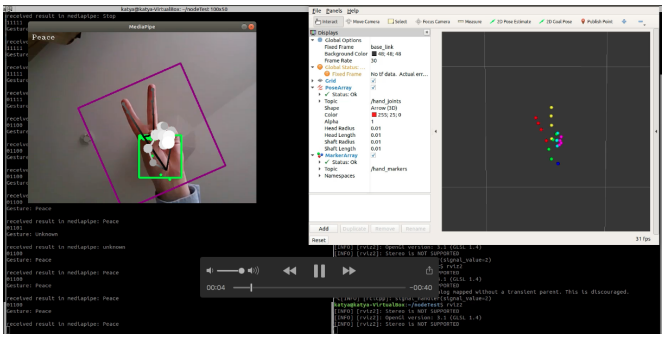


Fig. 11: Good lightning and uniform background (white wall)

2. In the good lightning conditions with a lot of objects on the background algorithm was able to detect 84 out of 100 gestures showed in the random order, which corresponds to 84% (fig. 12).

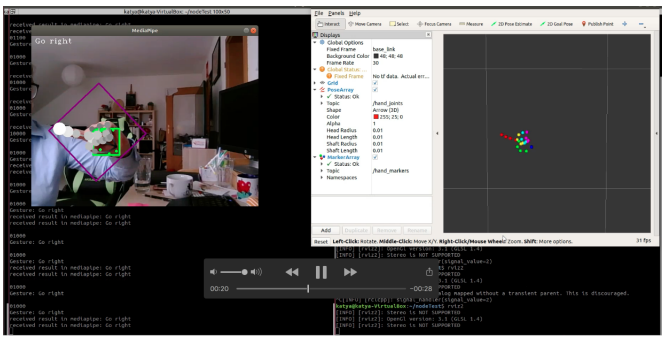


Fig. 12: Good lightning, but the background with a lot of objects

3. In the conditions of the bad lightning, but clean background the algorithm performed with almost the same accuracy (86%), as in the good lightning, but messy background (fig. 13).

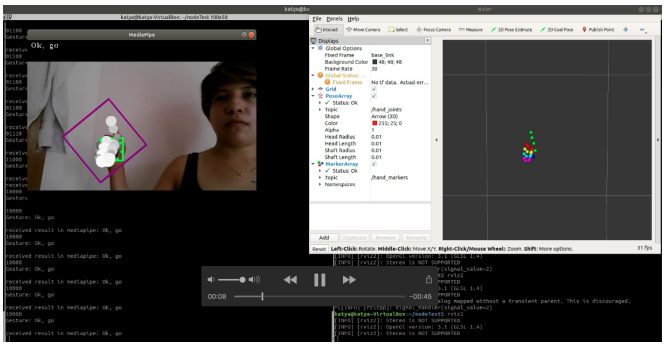


Fig. 13: Bad lightning, but uniform background

4. And finally in the conditions of the bad lightning and the background with a lot of different objects the algorithm showed 80% accuracy. In this specific case there was additional noise: some random objects, which were not hands were detected as hands, which did not affect the correct detection, but added noise.

The resulting accuracy for different scenarios can be seen in the figure below (fig. 14).

		lighning	
		good	bad
background	uniform	90%	86%
	a lot of objects	84%	80%

Fig. 14: The resulting accuracy for different scenarios.

5 Conclusions

For future work we see following improvements:

1. The geometrical relationship between joint of the fingers can be used more actively, for example, angles between different joints can be calculated for more precise gesture detection.

2. Alghought the Mediapipe library allows the multi-hand tracking, this functionality was not utilized in the gesture detection algorithm, so as future improvement of the project complex gestures, which consist of combination of two hands can be implemented.
3. The number of possible gestures in the current implementation is 2^5, which is 32 possible gestures one can come up with, alghought some of them would be harder to show physically, for example, when ring finger has to be closed while other fingers are open.

References

1. Abhishek, K.S., Qubeley, L.C.F., Ho, D.: Glove-based hand gesture recognition sign language translator using capacitive touch sensor. In: 2016 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC), pp. 334–337 (2016)
2. Köpüklü, O., Gunduz, A., Kose, N., Rigoll, G.: Real-time hand gesture detection and classification using convolutional neural networks. In: 2019 14th IEEE International Conference on Automatic Face Gesture Recognition (FG 2019), pp. 1–8 (2019)
3. MediaPipe: Mediapipe. https://mediapipe.dev/
4. Molchanov, P., Yang, X., Gupta, S., Kim, K., Tyree, S., Kautz, J.: Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural networks. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4207–4215 (2016)
5. Panwar, M.: Hand gesture recognition based on shape parameters. In: 2012 International Conference on Computing, Communication and Applications, pp. 1–6 (2012)
6. V. Bazarevsky and F. Zhang: On-device, real-time hand tracking with mediapipe. https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html (2019)
7. Vishwakarma, D.K.: Hand gesture recognition using shape and texture evidences in complex background. In: 2017 International Conference on Inventive Computing and Informatics (ICICI), pp. 278–283 (2017)
8. Wen, R., Yang, L., Chui, C., Lim, K., Chang, S.: Intraoperative visual guidance and control interface for augmented reality robotic surgery. In: IEEE ICCA 2010, pp. 947–952 (2010)
9. Xu, P.: A real-time hand gesture recognition and human-computer interaction system (2017)