Autonomous Robot Movement Control Through Hand Gesture Recognition

Yirong Tang, Shucheng Tian, and Shukun Zhang

Abstract

Intelligent gesture control brings a substantial improvement to natural Human-Computer Interaction. This Project aims to control the motions of Turtlebot through hand gesture recognition. This article presents a real-time algorithm that solves the problem of hand gesture recognition based on video inputs from 2D USB camera. The algorithm mainly consists of three steps: hand detection, feature extraction and gesture classification. For hand detection, we used a combination of utility functions from the open source OpenCV library to find the contour of the hand present in the image. Then we extract features of hand such as the number of convexities and number of finger tips. The features collected will then either be used as information for basic finger number recognition, or as input vectors for a multi-class SVM classifier to categorize complex hand gestures, depending on the user's choice. When tested in a controlled environment, the recall of both simple finger recognition and SVM classifier are steadily in the high ninety percent.

1 Introduction

The goal of Human Computer Interaction (HCI) is to bring the interaction performance similar to human-human interaction. One of the main components of human-human interaction is the use of gesture to convey information. Among the various body gestures, hand gesture is the most natural, intuitive, and versatile gesture.

Hand gesture recognition techniques utilized in past researches fall mostly into the following two categories: glove based method and vision based method. Glove-based method requires the human user to wear some sorts of device on their hand, and with wires that links the device to the computer. Vision based method, on the other hand, uses image processing techniques to extract the hand pixels from input sources such as video feed from the webcam.

Our program is based on Vision method. We feed the video from the webcam to a number of image processing functions from the open-source OpenCV library, and extract the hand from the background. Based on the gesture of the hand, the program is able to issue corresponding command to the turtlebot.

2 Related Work

In recent experiments using glove-based method [2], users had to wear heavy gloves with wires that links the gloves to the computer. For detecting hand gestures, some optical or mechanical sensors, actuator and accelerometer are attached with the glove. This approach forces the user to carry a load of cables which are connected to the computer and thus hinders the ease and intuitiveness of the user interaction, as well as demands more maintenance due to the complex wired structures[4].

On the other hand, vision-based method is more natural. However, in the past researches, recognizing gestures through a camera involves many aspects such as motion modeling, motion analysis, pattern recognition and machine learning[4]. The open source openCV library, designed for computational efficiency and with a strong focus on real-time applications[1], provides many image processing functions to programmers who need to handle such complex tasks, as well as machine learning algorithms to improve the autonomy of a program. In recent studies [5], hand gesture control mechanisms are divided into two steps: hand detection and gesture recognition. The most usual hand detection method is done by applying image processing techniques: the source images are converted into a black-white image (by grayscale, blur, and threshold filter), in which the hand is differentiated from the background. The features (largest and smallest contours, hull points, and convexity defects) computed from the 2D array of binary pixels are used to set classifiers and recognize gestures.

Recently, various machine learning algorithms and approaches are involved into gesture classification and recognition. Regarding the limits of pre-defined gestures, programmers are more willing to let the machine learning gestures by itself, supervised or unsupervised. In recent researches[6], the hand recognition approaches with machine learning algorithms (such as multi-layer perceptron, decision trees, and naive Bayes), have accuracy of more than 97%. Another research[3], Real-Time 3D Hand Gesture Recognition from Depth Image, suggests that their machine learning algorithm used in gesture recognition is support vector machine (SVM), which is employed to classify the shape of hand into different categories.

3 Methodology

3.1 Problem Formulation

The task of hand gesture recognition entails several challenges, including, but not limiting to, separation of hand pixels from background pixels, categorization of different hand gestures, and execution plan of the recognition-reaction loop(shown as the flowchart in Section 3.2).

To simplify the question at hand and for better performance purpose, the program operates under the following assumptions:

- The environment in which the application will be run is relatively consistent in terms of background pixels and lighting conditions.
- The trajectory of the hand is continuous at any given time T_0, and the gesture displayed by human user will be relatively static within a single recognition-reaction loop.
- Only one hand will be present in the image at any given time T_0, and that the hand is not too far away from the USB camera.

3.2 Technical Approach

3.2.1 Hand Detection

The problem that needs to be solve can be divided into two sub-problems, namely how to detect hand and how to recognize the gesture. In order for a hand to be successfully detected in a image, the image must first undergo a series of pro-processing procedures. The image processing functions are all defined in the OpenCV2[1] Library. For any given frame, the image is first converted to a gray-scale format using the **cvtcolor()** function, and then a Gaussian Blur of size nineteen by nineteen is applied to the gray-scale image using the function **GaussianBlur()**. Once the image is smoothened, the **threshold()** function will further reduced the unwanted pixels in the image, leaving mostly pixels that belong to the palm in the image. Once we have the result from the **threshold()** function, the result will be passed to the Hand Recognition Section.

3.2.2 Hand Recognition

In order for the program to recognize hand gestures, we implemented two algorithms. The simpler one of the two takes an approach called the convex hull and defect point method(CHDP).

Before we get into the details of the CHDP method, a few terminologies need to be defined. First, a contour is the outline of a certain object. A convex hull is defined as the smallest closed polygon that covers then entire contour, as shown here as the red polygon in Figure 1. A defect point is define as a point that is in the convex hull but not on the contour. The red points here in Figure 1 are the deepest defect points.

[1] https://docs.opencv.org/2.4.13.6/

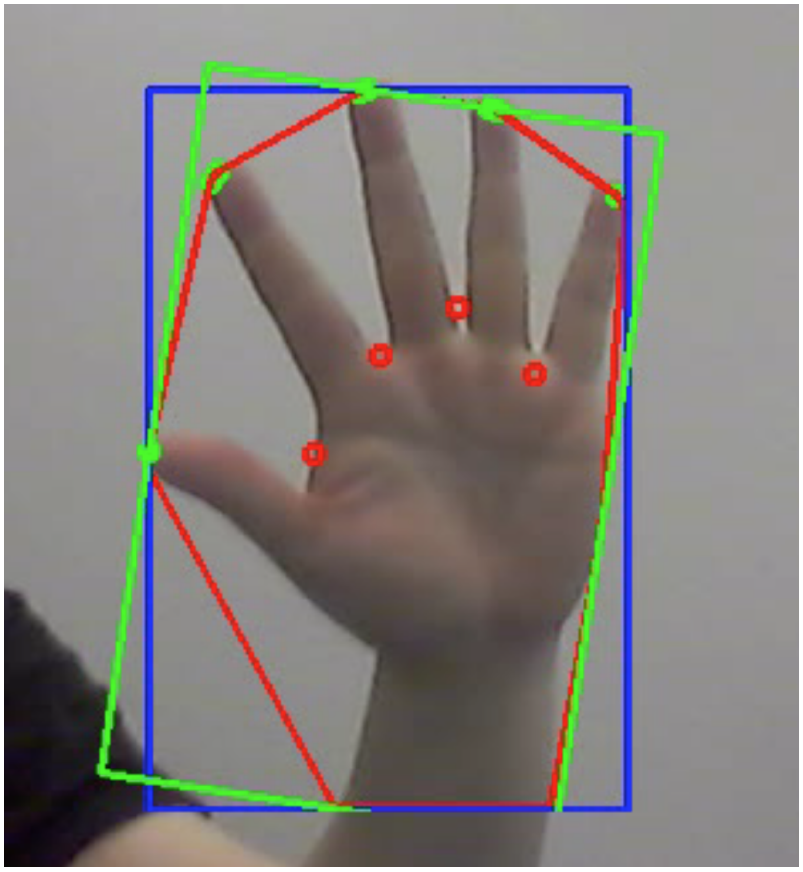

Figure 1: A example of hand detection in the program with visual aids.

The idea behind this convex hull and defect point method is that when a hand is present in the image, it will have a distinctive convex hull with a few large defect depth, which occurs at the valleys between two fingers. And the relationship between defect numbers and finger counts is quite simple: one defect is two fingers, two defect is three fingers and so on. However, this method does not distinguish a fist from one finger, because when the threshold value for defect detection is high, the defect here is not valid.

For the machine learning approach, we implemented a multi-class SVM classifier to categorize complex hand gestures whose differences may not necessarily lie in the number of fingers alone. We chose this classifier because (1)It is much faster to train than the neutral network models such as CNN and DNN, and (2) For the number of features and training data at hand, the SVM classifier achieves satisfactory results without exhausting the system resources or time.

For the SVM classifier, we came up with five fifty by three matrices as training data. Each matrix correspond to one gesture, and within each matrix, there are fifty data entries that are represented as triplets. The triplets contains three feature for a given hand:

1. The number of defect points(above a certain threshold)
2. The number of finger tips
3. The euclidean distance between the finger tips

Together with the built-in Kernel functions for SVM and the **CvSvm::train()**, we can build a multi-class classifier that can later be called through the function **CvSvm::predict()**.

3.2.3 Program Control Flow

The flow chart below gives a high level overview of the execution loop of the program. To achieve real-time hand tracking, the program takes in the video feed and process the video on a frame by frame basis. For every frame, the functions discussed in Section **3.2.1** are applied to the image. If the resulting image contains a valid hand contour, whose contour area falls between the upper

and lower limit of the threshold and is the largest contour found in the image, then methods in Section **3.2.2** are used to extract useful features from the hand contour. If machine learning method is chosen by the user, then the program will take $5 * 50$ screen shots, and calculate the feature matrices for the SVM classifier; if the CHDP method is used, the program calculates the most frequent finger number detected over 20 frames, and give that number as output for the robot control signal.

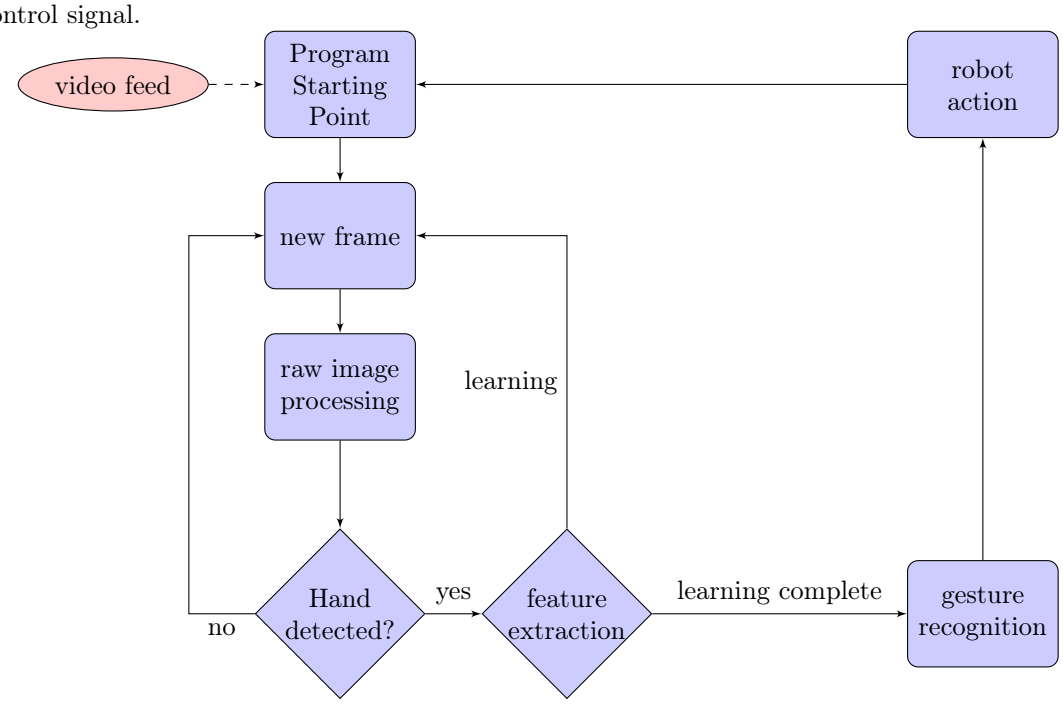

Figure 2: Control Flow of the Entire Program.

When the training of SVM completes, the program will signal the user that he or she can now give the gestures and the program will run that gesture through the classifier, and tell the robot to act accordingly. The actions of the robot are relatively simple, but it does not have to be simple, and can be altered easily with a few lines of code that publishes instructions to ROS topics of interest[2]. For our program, taking the CHDP method as example, two fingers would be for the robot to move forward, three fingers for moving backwards, four fingers for a 360 degree turn, and five fingers for moving forward and turning left or right, depending on the hand orientation.

4 Experiments and Results

4.1 Example Run

To run our program, open the terminal and launch the turtlebot and then the program itself from command line. When prompted, enter 0 on command line to choose default mode(1 for learning mode, which operates in almost the exact same steps except the program learns the gestures first before trying to recognize them). The user will then see a welcome message shown below.

[2] http://wiki.ros.org/Topics

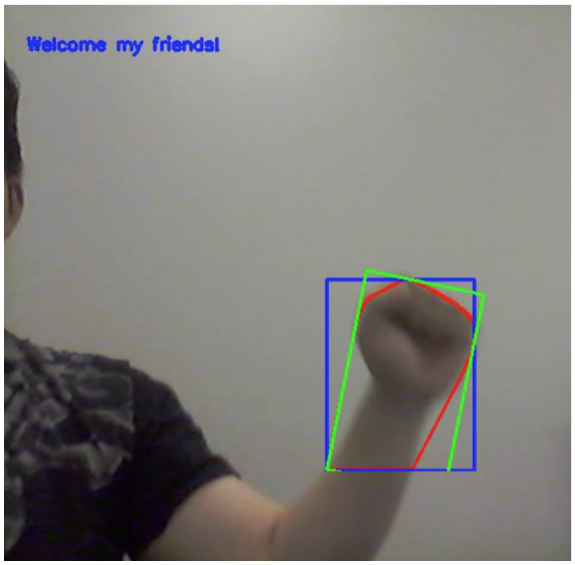

Figure 3: Welcome message at the start of the program.

Then the user should give a fist to the camera as a signal of ready to perform gestures, as shown in figure 4.

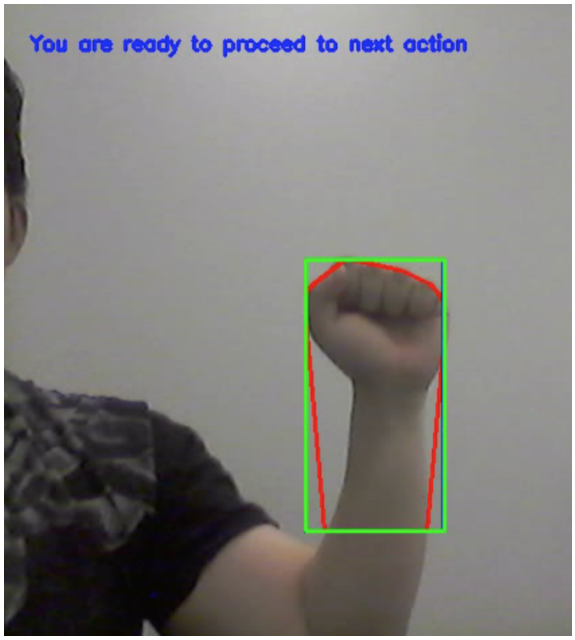

Figure 4: Give a fist to signal the next gesture for recognition.

Once the user sees that the program is ready, the user can give one of the five pre-defined gestures. Figure 5 shows the user giving two fingers, which tells the turtlebot to move forward. After the robot finishes his movement, the user should give a fist again to show him or her that the program is now ready for the next gesture. In fact, regardless of the mode the program is in(default or learning), the user should always give a fist and make sure that the program prompts "You are ready to proceed to next action" before giving the gesture for the program to recognize.

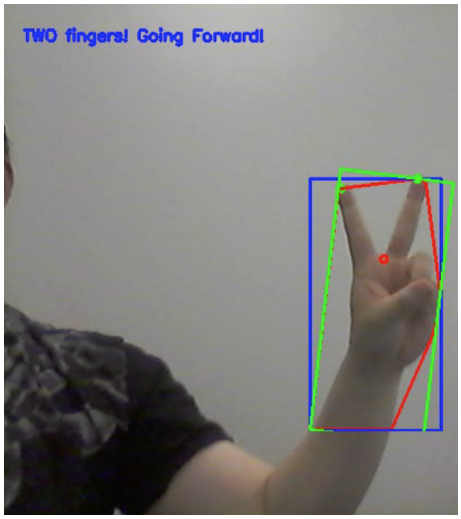

Figure 5: The robot sees two fingers and is going forward.

4.2 Quantitative Results

4.2.1 Ideal Environment

We first did our experiment in ideal environment, a light plain background free of noise. We gave a set of 5/10/20/40 gestures[3]and figure 3 shows the number of correctly recognized hand gestures.

Correct recognize gesture in learning mode	4	9	17	36
Correct recognize gesture in default mode	5	10	20	39
Number of gestures given	5	10	20	40

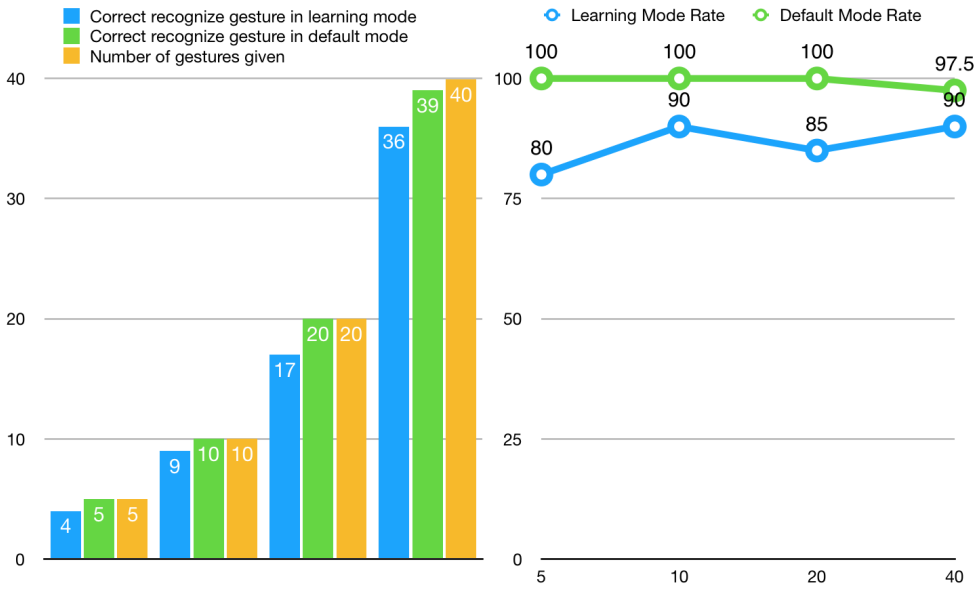

Figure 6: Results from default(CHDP) mode and learning(SVM) mode in ideal environment.

[3]For learning mode, the numbers represent the number of trials run on all five gestures learned from the user.

4.2.2 Complex Environment

We then try our experiment in a random environment, which is a complex background with noise. We gave a set of 5/10/20/40 gestures[4]and figure 4 shows the number of correctly recognized hand gestures.

Correct recognize gesture in learning mode	2	4	8	15
Correct recognize gesture in default mode	3	5	11	25
Number of gestures given	5	10	20	40

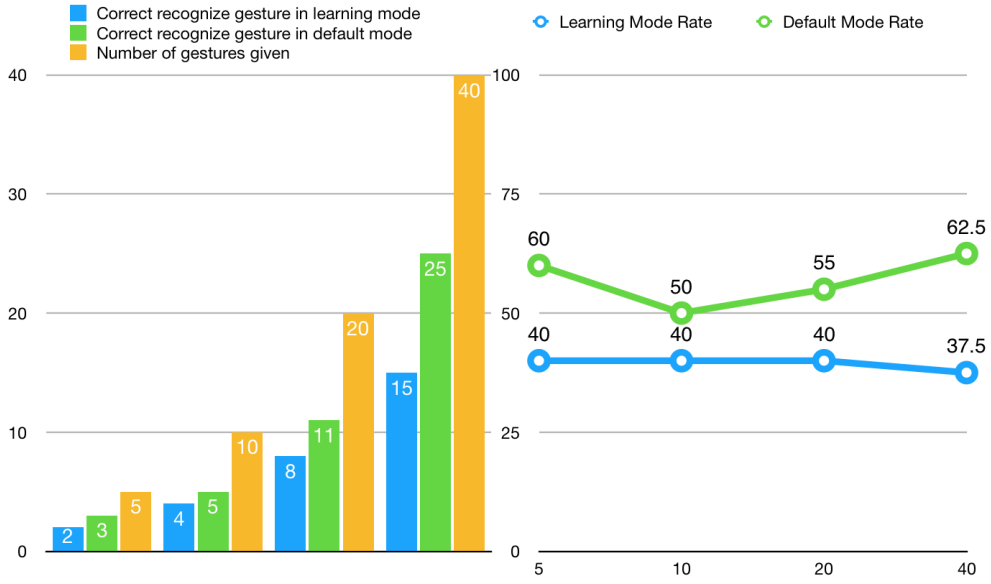

Figure 7: Results from default(CHDP) mode and learning(SVM) mode in complex environment.

4.3 Discussion

4.3.1 Image Background

We assume the environment we used in computing vision is ideal, which means the environment is simple and has little noise. However, in a complex environment, the robot could not recognize such hand gestures, whether in default mode or learning mode. The reason is: the method we used to detect hand gestures is based on the contour detection, which is based on the threshold operation on images. If environment has too many noises, the threshold cannot separate the hand from background. However, if the hand's pixel intensity are exceptionally consistent(i.e. wearing a red glove), it is doable.

4.3.2 Default (CHDP) Method: Finger Detection

For finger detection in default mode, we use defect points: convexity defect = MAX(depth for D_0) > THRESHOLD. For now, the threshold is low. If we increase the number, the accuracy will be high, but at the same time user should be closer to camera to perform hand gestures. If the number is low, the user can be comfortable when he performs, but at the same time, the angle between plan and little finger will also be considered to a defect point.

To improve efficiency and accuracy in default mode, we compromised on user-friendliness to perform well on two to five (could be higher than five) finger detections, but we cannot distinguish no finger and one finger, since the depth of the defect created by the one finger and the palm is lower than our threshold value.

[4]For learning mode, the numbers represent the number of trials run on all five gestures learned from the user.

4.3.3 Learning Mode

4.3.4 Assumptions

4.3.4.1 2D Projection Similarities

Although we assume that the program will run in a ideal background, there are still three other assumptions for learning mode to perform well. First, the different gestures given by users should not be same, or almost same, in a 2D image. Since we use a 2D camera for feeding the video and do threshold filtering to convert the images to black-and-white pixels, if the contour of the hand projected on the 2D image is same, the program could not separate these gestures.

4.3.4.2 Orientation of hand gestures

Since the features we choose to train the data set are not enough to distinguish the gestures presented in different orientation, the user needs to give the gesture in exactly the same orientation as entered in the training data set, or the classifier will treat the input as an undefined gesture.

4.3.4.3 Restriction on Relative Position

The position of hand relative to camera is important in our assumptions. Because one of the three features we used in the contour area, same hand gestures shown on different distance from the camera will lead to different contour area. Since we use SVM as a classifier, different gestures are separated by linear plane. Same gestures with a big different in one feature will make a huge difference on the axis of that feature. As a result, the gesture could be placed in a faulty classifier and fail to recognize such gestures correctly.

4.3.5 Improvement

To solve above problems, we have several plans, but, considering the time limit of this project, we were not able to experiment and implement these plans, we will discuss this part in detail in the future work session.

5 Conclusion

5.1 Achievement

We demonstrate intelligence and autonomy in our project. We use Convex Hall Defect Point (CHDP) method to enable the robot to recognize a set of commonly used hand gestures; furthermore, we use machine learning to train our robot to recognize a wider set of gestures that is not predefined. The robot achieves a high success rate at gesture recognition and is able to move accordingly at all times in a simple environment, namely with a clear background. The ability to learn and react to visual input data (images from camera) demonstrates that the robot achieves a certain level of intelligence and autonomy.

5.2 Future Work

Although the robot has the ability to recognize the gesture once the hand gesture is clearly presented in the image, to improve the robot's performance and enable it to work in more complex environment, we must make it possible for the robot to obtain clear images of hand gestures in all possible scenario.

To achieve this goal, there are several approaches we can adopt. First, we can threshold the image within region of interest (ROI) instead of the entire image. By employing this strategy, we can effectively decrease the interference of the rest of image so that the robot can have a higher chance of successfully detecting a hand.

Second, after successfully detecting a hand in the region of interest(ROI), we can scale the image of the hand to a fixed size–scale up is the size of the image is smaller than the sized size and vice versa. Currently, the robot requires the user to keep his/her hand a certain distance from the camera throughout learning period and prediction period, because, except for the number of defects, both size of contour and Euclidean distance between finger tips is calculated based on the

absolute value. By adopting this strategy, we can produce more stable data for features and allow user to present gestures with a larger variance of distance from the camera.

A lot of other improvements can be made to increase the accuracy of gesture recognition and increase the set of different gestures that the robot can recognize. One of the most helpful improvement might be adopting 3D camera. By adopting 3D camera, we can gather a very important data from visual input–depth, and we can use depth to create more meaningful and useful features to better lean and differentiate various gestures.

References

[1] Gary R. Bradski and Adrian Kaehler. In *"Learning OpenCV: Computer Vision with the OpenCV Library."*, 2008.

[2] Ruchi Manish Gurav and Kadbe K. Premanand. "real time finger tracking and contour detection for gesture recognition using opencv". In *International Conference on Industrial Instrumentation and Control (ICIC)*, pages 974–977, 2015.

[3] et al Lin, Song. Real-time 3d hand gesture recognition from depth image.". In *Advanced Materials Research. Vol. 765.* Trans Tech Publications, 2013.

[4] G. R. S. Murthy and R. S. Jadon. "a review of vision based hand gestures recognition.". In *International Journal of Information Technology and Knowledge Management 2.2*, pages 405–410, 2009.

[5] et al Qureshi, M. Ali. "implementation of an efficient algorithm for human hand gesture identification.". In *Electronics, Communications and Photonics Conference (SIECPC)*. IEEE, 2011.

[6] Shankar Khatri Zuberi, Farooq Ahmed and Khurum Nazir Junejo. "dynamic gesture recognition using machine learning techniques and factors affecting its accuracy.". In *Innovative Computing Technology (INTECH)*. IEEE, 2016.

Acknowledgement

We would like to acknowledge Professor Jivko Sinapov at Tufts University for providing us with the Turtlebot and advising us with regard to SVM training data preparation. We would also like to acknowledge Srijith Rajeev for giving us example C++ code that utilizes OpenCV functions. Finally, we would like to give special thanks to Kaibei Lin, Peixuan Wang, and Chengye Yin for their unconditional support along the way.