Authentication by Hand Gesture

Quan Tong

Computer Security Final Project

**Introduction**

Sign languages are powerful as they can be shared by people who use different verbal languages. In this project, sign language will be used as an authentication. A series of hand gestures will play a role as a passcode to gain access to the machine.

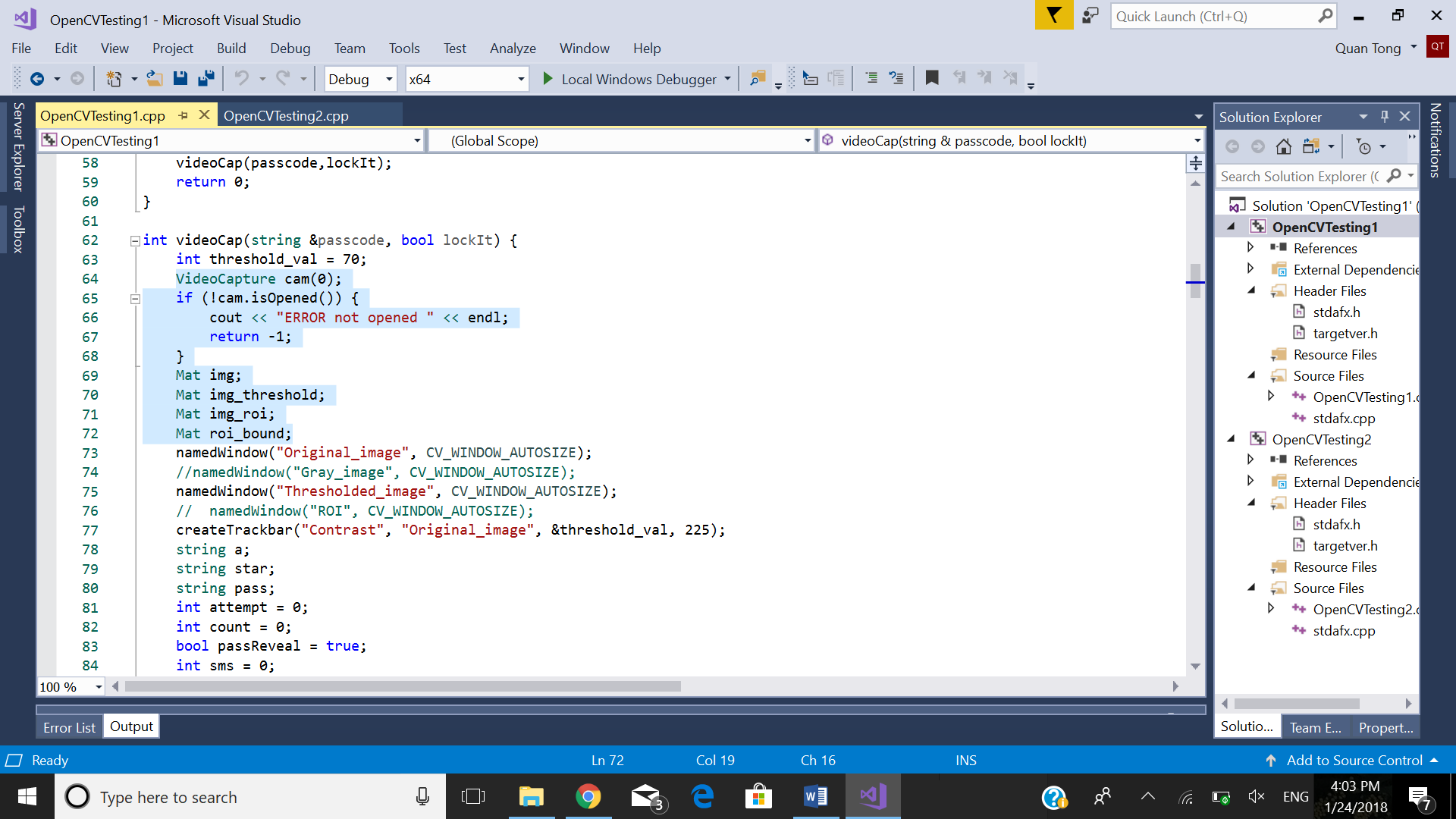
**Background**

Nowadays, authentication has been improved for more safety. Each authentication method has its own pros and cons, therefore, to enhance the accuracy of authentication, we use multi-factor authentication so one authentication method can back the other up. For example, bank accounts and many other website accounts use private questions as a backup authentication to validate a person’s identity. iPhone X and S8 both have fingerprint and facial recognition as the ways to authenticate oneself. Password is the backup solution if fingerprint and facial recognition do not work that can be due to several conditions. Therefore, hand gesture recognition can be added to this category. It can stand alone as a way to authenticate oneself or can be a part of a multi-factor authentication. Acknowledging its pros and cons, authentication by hand gestures can be used appropriately.

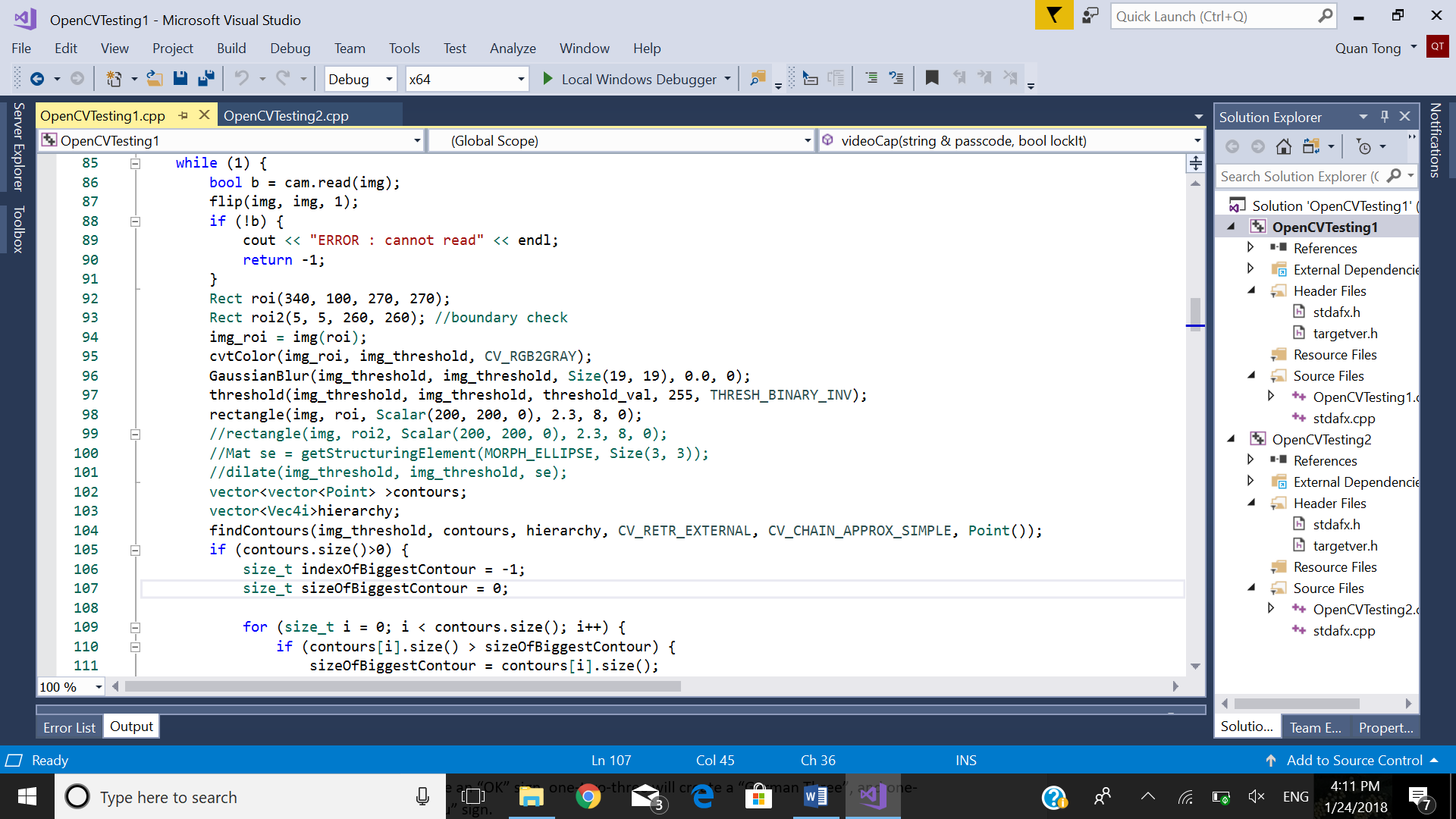
Authentication by Hand Gesture has two components in general. The first component is the authentication system. Specifically, the authentication must be able to lock and unlock the machine as a minimum requirement. The second component is the hand gesture recognition which is the complex component and also the most interesting part of the project.

**Materials and Methods**

This program is written in C++ and uses OpenCV, a well-known library of programming functions mainly aimed at real-time computer vision. There are 6 steps of implementation.

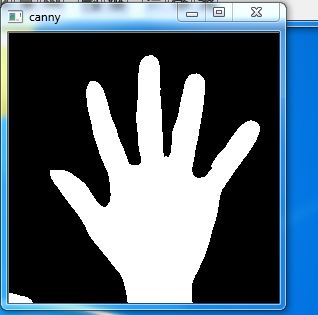


The second version of OpenCV is compatible with Windows 10. The first version of OpenCV is not recommended because there are many outdated features. We can open the camera or webcam

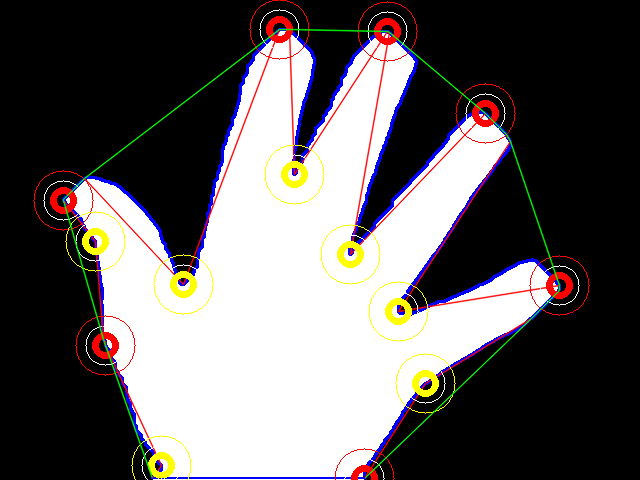


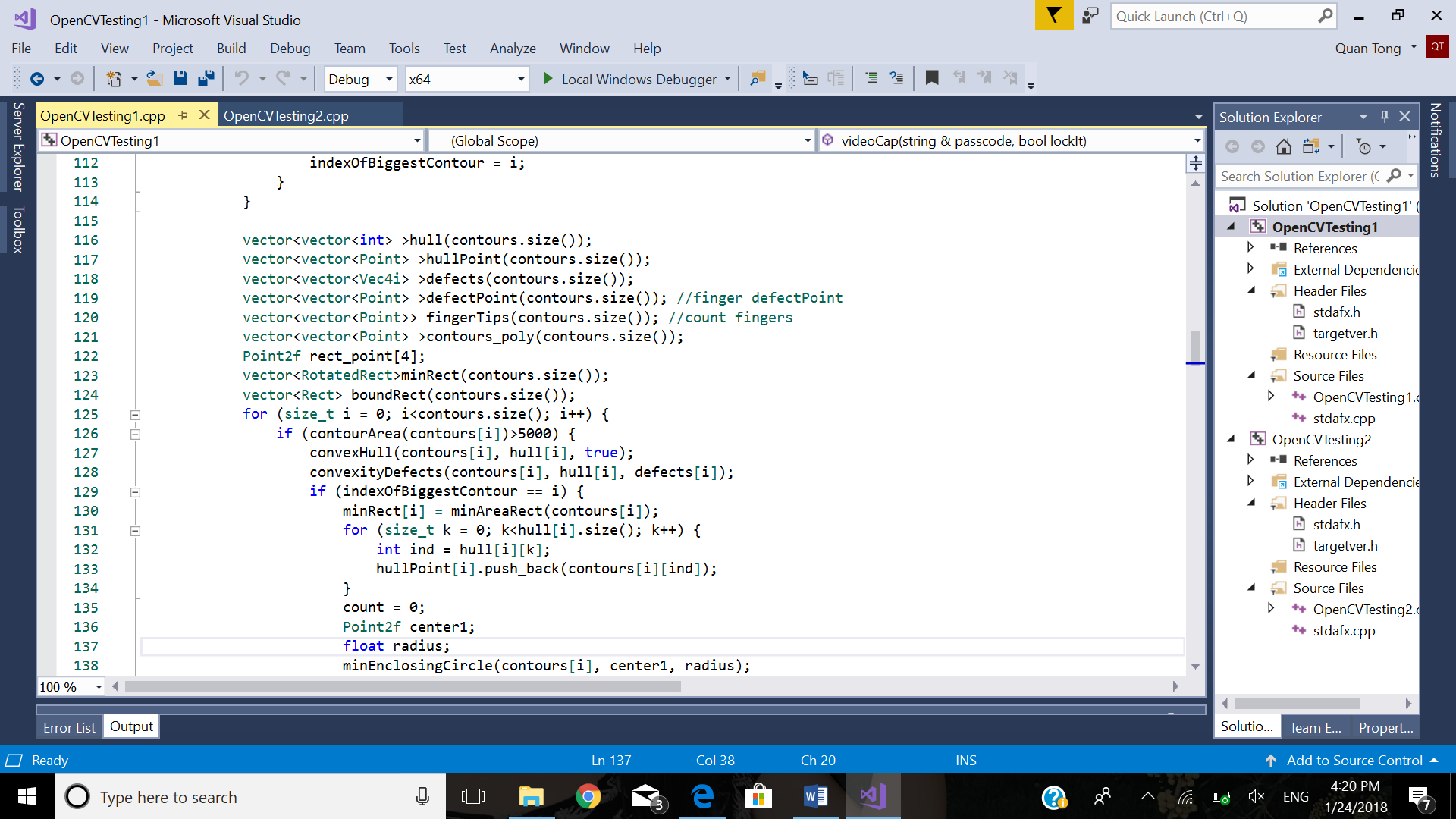
The first step is manipulating the image to subtract background as much as possible. I applied a simple method that contains 3 procedures: **cvtColor** that is used to produce a grayscale copy of the image, **Gaussian Blur** that blurs the image using Gaussian filter, and thresholding method that is used to get the binary image. We also want to narrow down frame of getting hand gestures to one part of the screen so the sensor won’t get confused because other objects. Since the background is not always fully subtract depending on lighting condition, I created the contrast taskbar that the user can adjust to subtract the background as much as possible.

[](https://cloud.githubusercontent.com/assets/9850882/12191826/175c214c-b5ff-11e5-8053-1a9b025de1bd.JPG)[](https://cloud.githubusercontent.com/assets/9850882/12191827/1941861e-b5ff-11e5-9a06-5182a3c4a990.JPG)

[](https://cloud.githubusercontent.com/assets/9850882/12191825/117deab2-b5ff-11e5-9a7d-3620092ce69d.JPG)

After getting a clear image of the hand, the second step is creating the convex hull and convexity defects for hand recognition. Finding the points of the convex hull and convexity defects are supported by **OpenCV** functions **convexHull()** and **confexityDefects().** These functions’ details can be found on OpenCV website. Below is an instance of the implementation.

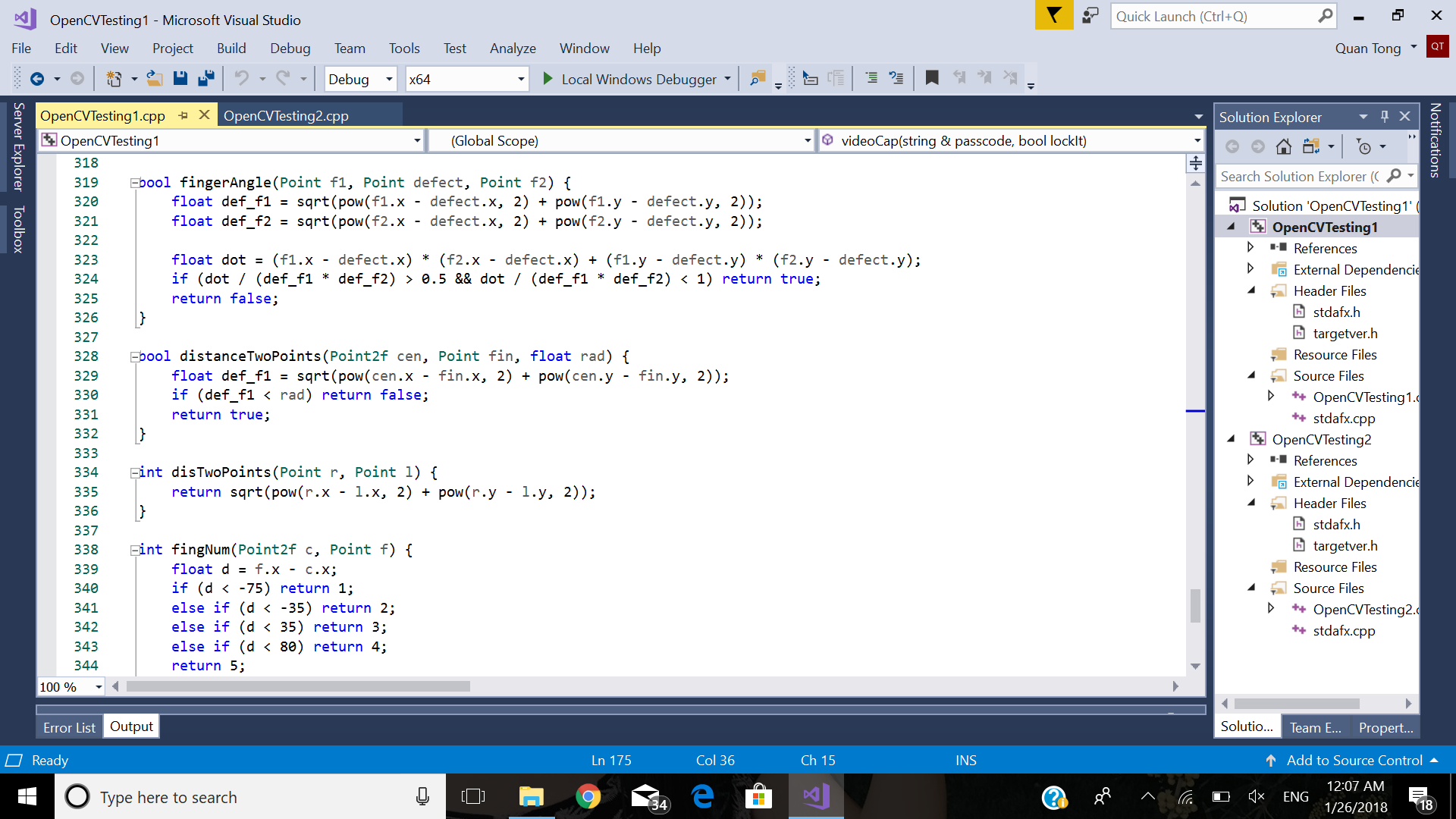




From all the hull points, I determine the most recognizable hull points, which are my fingertips or the joints between my fingers and my hand, to form a hull of my hand. The array of array of defects are all the defects in my hull, and I will have to accurately choose the one defect point that illustrate the gap between my two fingers. This one is the most crucial part because if I get a wrong defect point, I will get the number of fingers wrong. For example, if I have 5 hull points and 4 defect points, I definitely have 5 extended fingers. However, if I have 5 hull points and only 3 defect points, then I only have 4 fingers. One of the hull points is not my fingertip.

The third step is finding the center of the hand palm so we have a more accurate picture of the hand. Based on the distance between the convex hull vertices and the center of the hand palm, we can determine which vertices are fingertips and which vertices are only a part of the palm or adducted fingers. Through this, the number of extended fingers are counted and shown on the screen by **OpenCV** basic drawing functions.

The forth step is calculating the angle of the fingers and the horizontal line cutting through the palm. For example, middle finger is likely to make a 90-degree angle with the horizontal line and thumb is likely to make 45-degree angle. By finger distinction, we are now able to recognize the hand gestures.

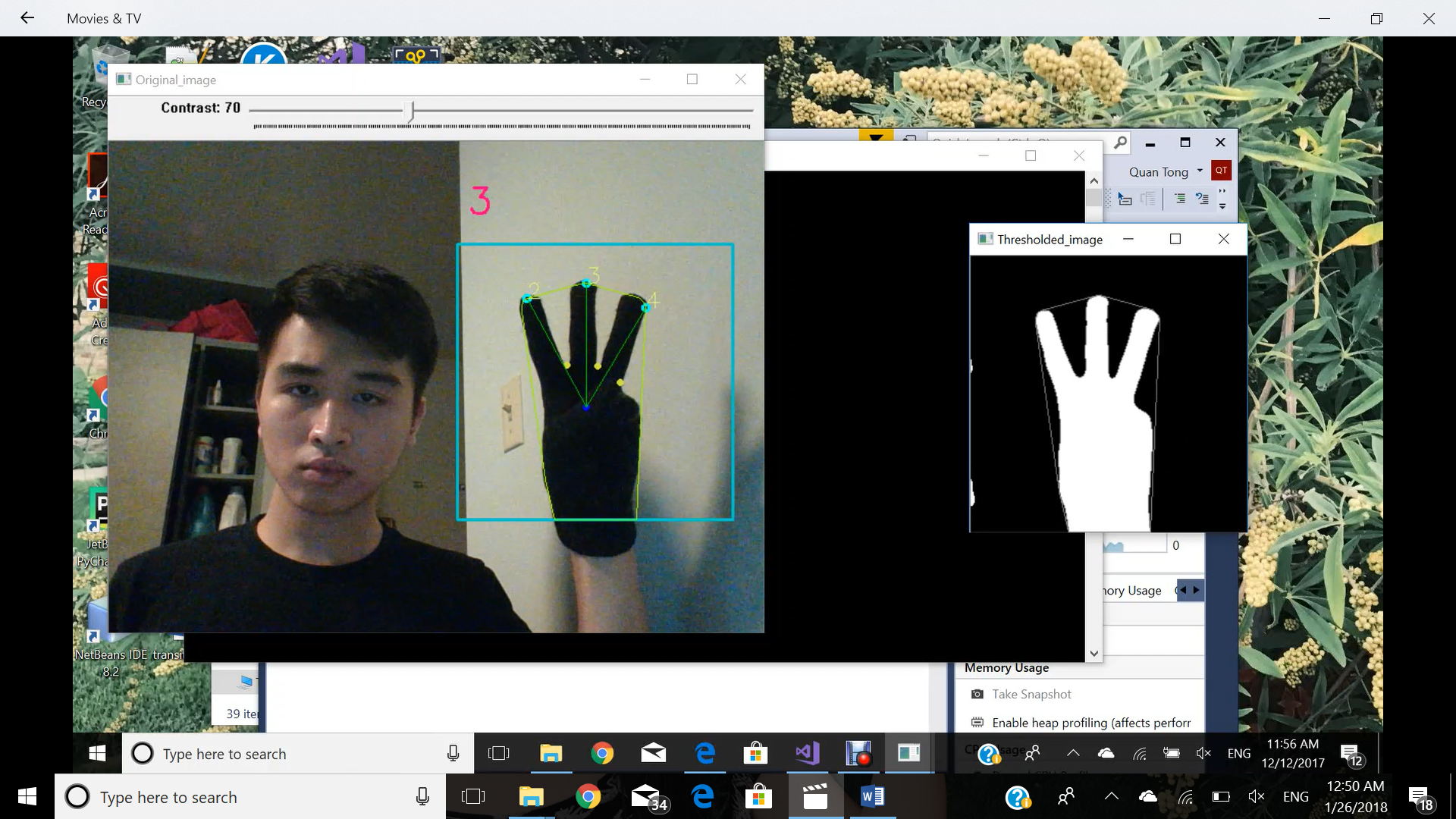


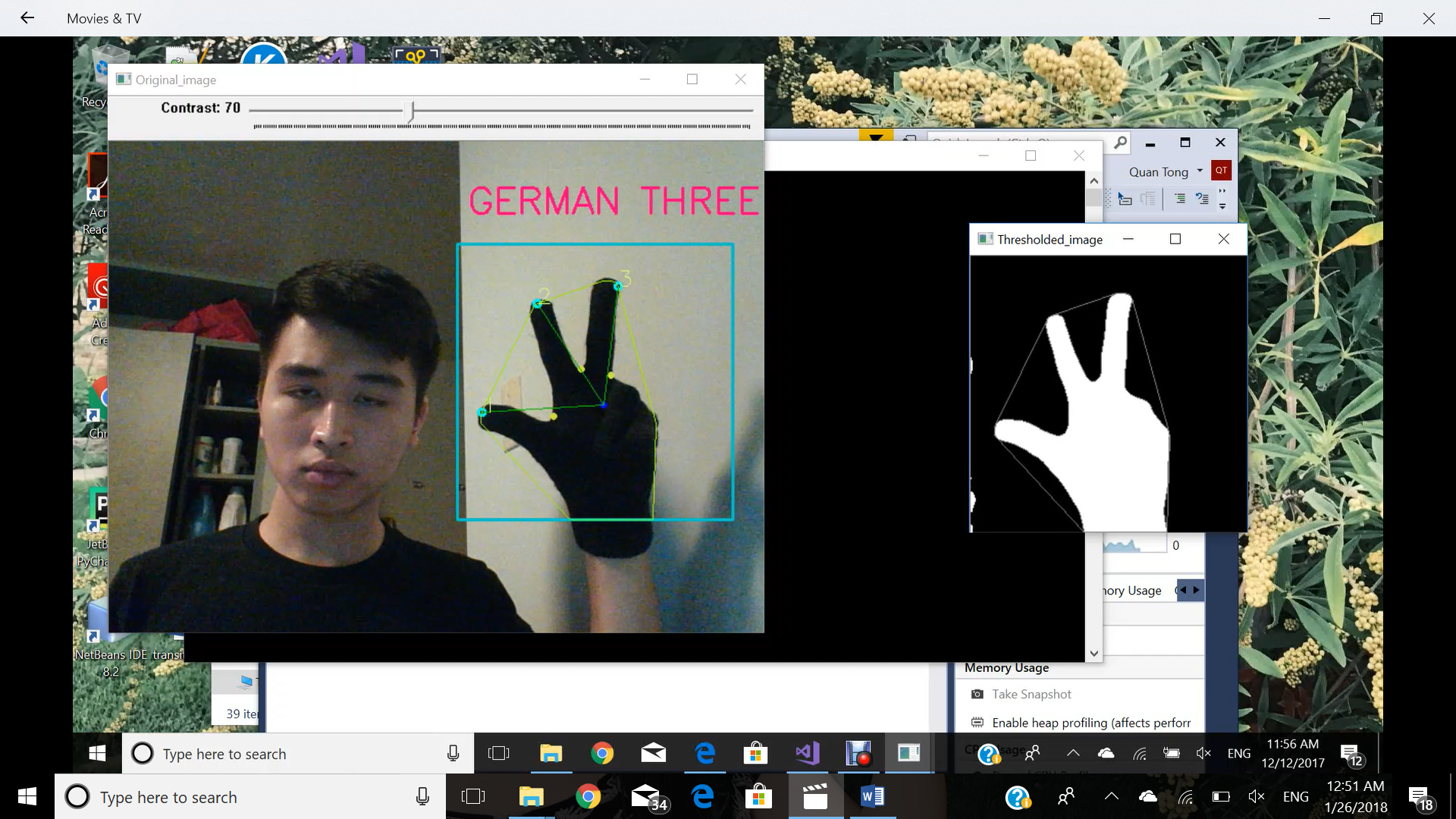
The fifth step is creating a set of hand gestures based on the number of extended fingers and the finger distinction. **fingNum()** returns the finger number. For example, the number of extended fingers are three. Counting from 1 to 5 as thumb to pinkie, three-four-five will create an “OK” sign, one-two-three will create a “German Three”, and one-two-five will create a “I Love You” sign.

In the last step, I create an authentication system that allows user to create a series of hand gestures as a password to lock the machine. If a user wants to unlock and access the machine again, the user must provide the same series of hand gestures. If the user takes more than four tries, the system will be suspended for 5 seconds.

**Result**

It is stable during the recognition of all available gestures. There are 10 gestures in total. The unrecognized gestures will be put as the number of extended fingers. It works more accurately with a black glove and some certain types of hands. Here is the footage of the program.





**Discussion**

The limitations of this program including the inability of recognizing hand colors, hand types, position of the hand in front of camera, and only the basic gesture set can also be the expansion of this program in the future. By applying machine learning, it is possible for the machine to learn more gestures and to improve the accuracy in recognizing gestures. For example, if the machine learns a person’s hand structure and how that person does a gesture, it will be able to determine if this is the right gesture being done by the right person. The database of hand gestures can not only be enriched by the existing types of gestures but also be enriched by the category of hand types.

**Conclusion**

This program can be the foundation of hand gesture recognition, and the biggest improvement to this program can be done through machine learning and improvement in accuracy.

**Citation:**

OpenCV library:

<https://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/hull/hull.html>

<https://docs.opencv.org/2.4.13.2/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html>

<https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html>

<https://docs.opencv.org/2.4.13/doc/user_guide/ug_mat.html>

Consulted implementation:

<https://github.com/udit043/Hand-Recognition-using-OpenCV/blob/master/README.md>

<https://www.hindawi.com/journals/tswj/2014/267872/>

Basic hand gestures:

<https://www.pinterest.com/explore/illuminati-signs-and-meanings/?lp=true>

<https://kmtom.wordpress.com/2010/12/13/hand-signals-illustrated/>