**Object Detection/Recognition for Haofu**

**Goal :** *With a technician wearing a portable head camera, be able to recognize which tool he is carrying.*

**The process is separated into two parts :**

Contents

[1. Object detection 1](#_Toc487529846)

[Color: 1](#_Toc487529847)

[Image tracking : 2](#_Toc487529848)

[2. Image recognition 2](#_Toc487529849)

## Object detection

In order to detect the object carried, the hand must be detected. This is a background subtraction application, with a moving background. We wanted to combine two methods : **Color Selection** and Image Tracking

### Color:

1. Color selection
2. Contour detection
3. Selection of the contours touching the bottom part of the image (other contours cannot be the user's hand)
4. Selection of the biggest remaining contour
5. Approximation of the contour
6. Estimation of the position+size of the object according to the position+size of the hand.

|  |  |  |
| --- | --- | --- |
| Figure 1 : Original video (the colors change are of no importance) | Figure 2 : Color selection | Figure 3 : Contour detection/selection + poly approximation + center |

*Results :*

As this is only first testing, no quantification of the results has already been done. The observations following are purely qualitative.

The results are not so bad, however, of course, when objects of the same color appear, such as wooden furniture, the results are not very good, even with the contour selection.

**Hand/ Gesture Recognition:**

1. Segment the hand skin by detecting skin color
2. Get n biggest contours
3. Describe contours with convex polygons and find convexity defects
4. Dismiss contours not representing hands
5. Separate a certain area around the hand for further tool recognition

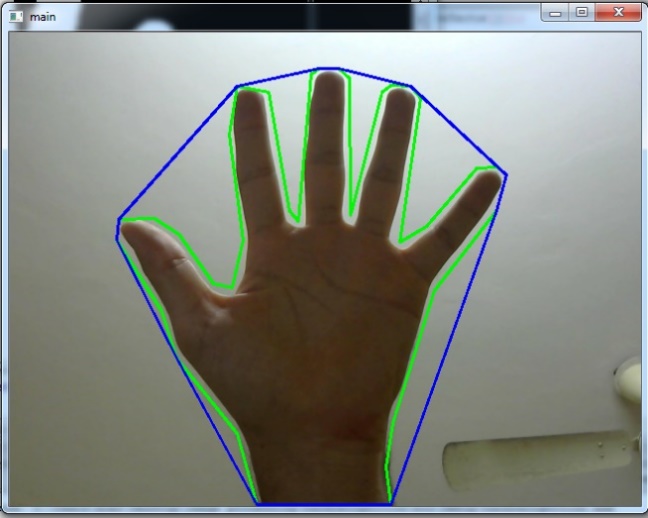


Figure 5: example of hand detection

*Results:*

It is the next step we are going to try, which is able to optimize the skin color detection when other objects have the similar color, so that we can get a more ideal focusing area for tool recognition.

### Image tracking :

Tracking important features in the video. The points belonging to the background should have a linked movement (rotation + translation). The other points belonging to the hand+object will have a different movement.

Separating the hand+object from the background.

*Using openCV Lucas-Kanade optical flow method.*

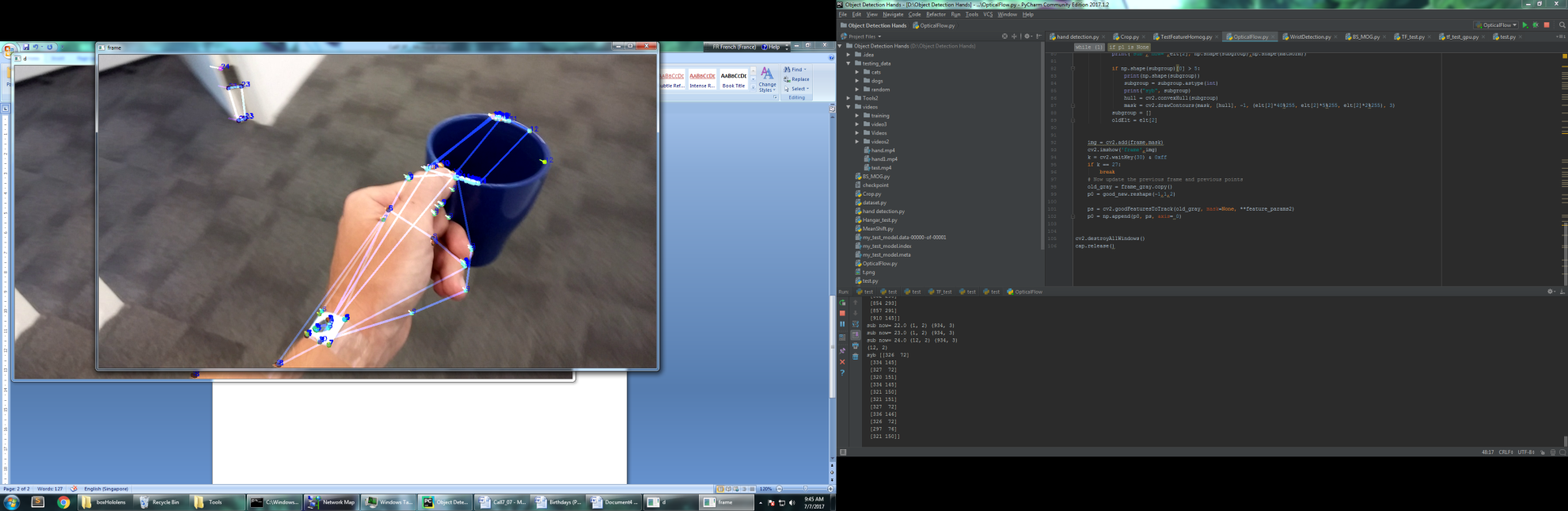


Figure 4 : Quick test connecting the points with same movement vector norm

*Results :*

Currently, we have no quantified results with this method, only because it is not finished. To continue, we need to :

1. Estimate the global movement of the camera (translation + rotation)
2. Separate the hand+object from the background

## Image recognition (Deep Learning)

**Part 1:**

First try using the open source library TensorFlow from Google. Adaptation of an online tutorial making the difference between cats and dogs, to try to differiencate 6 tools, using CNN.

The tutorial architecture is :

* 3 convolutional layers (the filter is a random normal distribution)(We tried adding and removing layers, the results were better with more layers)
* 1 flattening layer
* 1 fully connected layer

|  |  |  |  |
| --- | --- | --- | --- |
| D:\Object Detection Hands\Tools2\testing_data\blueScissors\blueScissors.1.jpg | D:\Object Detection Hands\Tools2\testing_data\boxHololens\boxHololens.20.jpg | D:\Object Detection Hands\Tools2\testing_data\greenScrewdriver\greenScrewdriver.21.jpg | D:\Object Detection Hands\Tools2\testing_data\pinceVerte\pinceVerte.39.jpg |

Figure 5 : Examples of the tools

**Process :**

1. We took some still videos of us holding different tools, on the same white background.
2. From the videos, we extracted the frames (tried with 200, 500 and 2000 images)
3. We fed all the data to the model, and trained it, with :
   1. Original Frames
   2. Frames after background subtraction
   3. Contour detection frames (we had the best results)
4. We tried the result on an other video showing every tools one after the other.

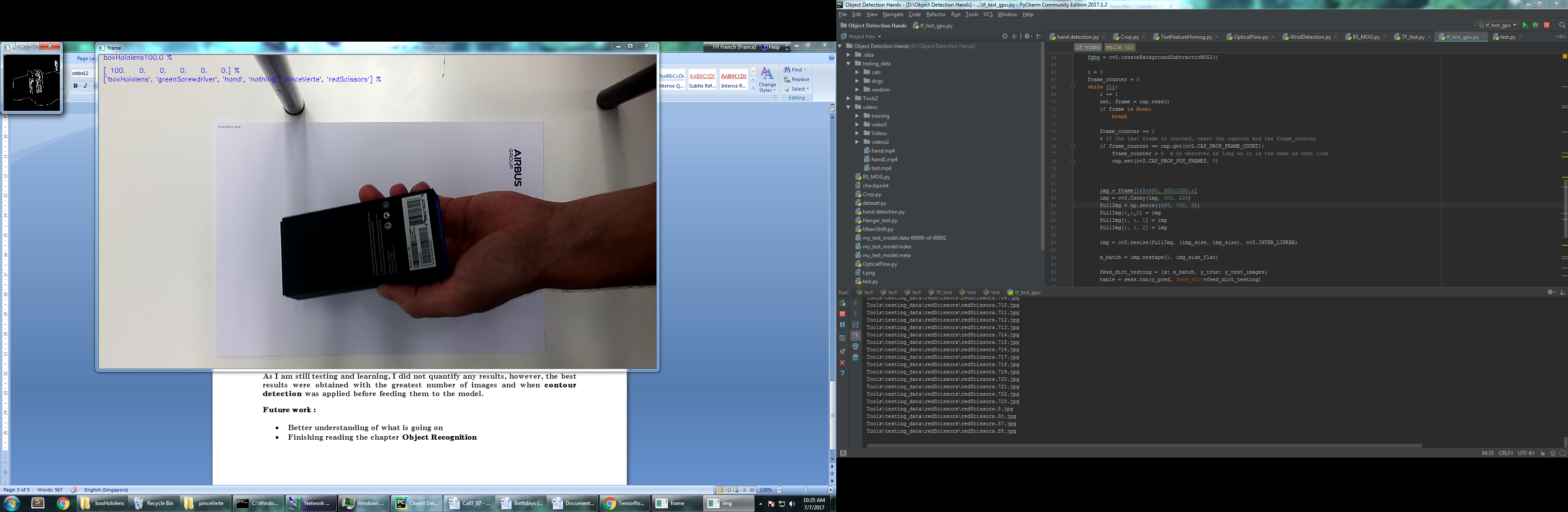


Figure 6 : Video test, the name of the object detected is in the top left hand corner

As we are still testing and learning, we did not quantify any results, however, the best results were obtained with the greatest number of images and when **contour detection** was applied before feeding them to the model.

Roughly speaking, 5 tools were detected out of 6, for the best model.

**Future work:**

* Better understanding of what is going on.
* Try to avoid overfitting, which might be the current issue.
* Build a dataset, using real and synthetic images.

**Part 2:**

**Concepts**

**Image classifier:**

* When no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

So:

* Data-driven approach:

1. Collect a dataset of images and labels

2. Use Machine Learning to train an image classifier

3. Evaluate the classifier on a withheld set of test images

**NN classifier (k-nearest neighbour) :**

* Remember all training images and their labels, and predict the label of the most similar training image
* Use distance matrix to compare test images and training images, find pixel-wise absolute value differences. The lowest is classified into the same class.
* Choice of distance: hyperparameter (eg. Manhattan distance, Euclidean distance).

How to choose: tuned using a validation set, or through cross-validation if the size of the data is small

* k is also a hyperparameter to show no. of classes
* problem: never used to classify images. Cannot detect for any shift, messed up or light adjustment.

10\*3072 3072\*1

---Example of MMUL

10\*1

**Linear Classifier:**

W: Weight

b: Bias

x: Determined by photo

**Loss Function:**

* **Hinge loss-- (Multiclass) SVM loss**

Given an example where is the image and where is the (integer) label, and using the shorthand for the scores vector

: score vector of the correct class

The larger the worse

**Score Function:**

**Full Training Loss**

: regularization strength (hyperparameter)

* **Cross-entropy loss-- Softmax Classifier** (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

Where

0.13

0.87

0.00

24.5

164.0

0.18

3.2

5.1

-1.7

exp🡪 normalize🡪 🡪

Provided the 1st class is the correct class

The closer to 0, the better.

**Optimization:**

Follow the slope. In multiple dimensions, the gradient is the vector of partial derivatives

The loss is a function of W

* Numerical gradient: approximate, slow, easy to write
* Analytic gradient: exact, fast, error-prone
* In practice: Always use analytic gradient, but check implementation with numerical gradient. This is called a **gradient check**.

**Back Propagation** All partial derivative

Loop:

1. Sample a batch of data

2. Forward prop it through the graph, get loss

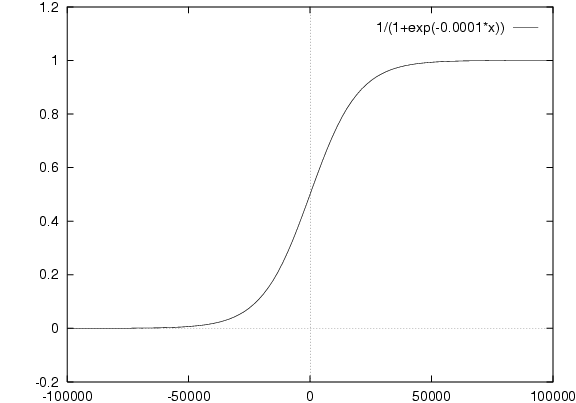
3. Backprop to calculate the gradients

4. Update the parameters using the gradient

**Activation Function:**

Ideal features: 1. Range [-1,1]

2. Zero centered

3. Doesn’t saturate

* **Sigmoid**

Problem: 1. Saturated neurons “kill” the gradients

2. Sigmoid outputs are not zero-centered

3. exp() is a bit compute expensive



* **Tanh(x)**

Good: 1. Squashes numbers to range [-1,1]

2. zero centered (nice)

Problem: 1. still kills gradients when saturated



* **ReLU**(Rectified Linear Unit)

Good: 1. Does not saturate (in +region)

2. Very computationally efficient

3. Converges much faster than sigmoid/tanh in practice