Computer Vision 2021/2022

Human Hands Detection and Segmentation

Chioccarello Stefano, Lacini Ralton, Novero Andrea
{stefano.chioccarello.1 , ralton.lacini , andrea.novero}@studenti.unipd.it

July 2022

Contents

1 Introduction

1.1 Activity goal

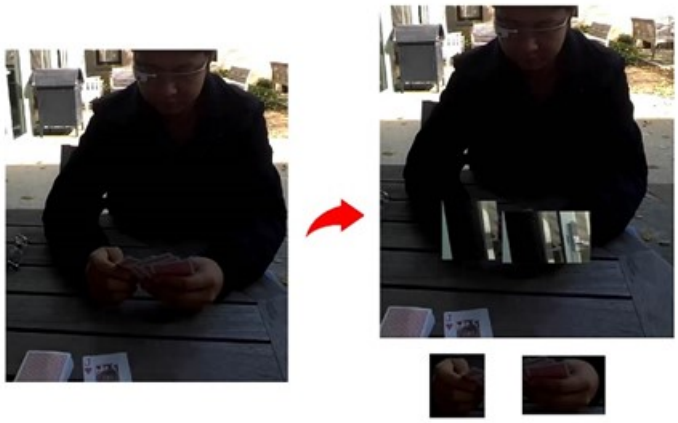
The aim of this project was to develop a system able to:

1. Detect human hands in an input image
2. Segment human hands in the image from the background (i.e. perform a binary segmentation)

The main difficulties of this task are given by many factors. Even though human hands share some common features, many elements can sensibly affect their appearance in the images, such as different skin tone, presence of clothes, occlusions when holding objects in the hand, different viewpoints, more than one hand in the same image etc.
The system had then to be tested on a set of test images to assess its performances and robustness. Such images come from two different datasets (EgoHandsHandsOverFace). The metrics used to assess the solution are: Intersection over Union (IoU) for the detection part; pixel accuracy for the segmentation.

1.2 Data acquisition

For both of our datasets we extracted the hand subimages by using the provided ground truths and saved them for later use. The rest of the image was then modified to contain some other section of the image where there were hands previously. These images were then used as negatives. During the whole process, the provided test images were removed entirely from the dataset. Another source of negative images was taken from the following dataset (Faces only dataset and folder "00" only). It then followed a laborious phase of manual inspection in order to cleanse the data from "bad" images. In particular we removed very small and out of focus images from the first two datasets and removed all pictures containing hands or parts of hands in the imdb dataset.

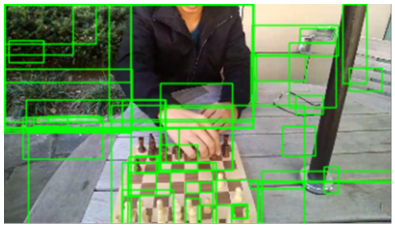
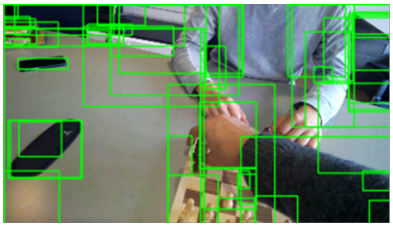
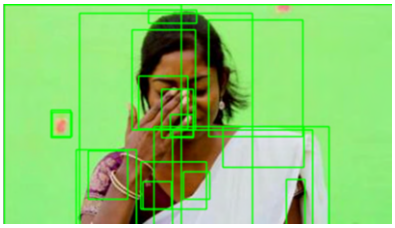


2 Hand Detection

2.1 Detection approach

First, we tried some low and mid-level object detection approaches, exploiting some pixel-level techniques rather than higher-level image processing algorithms, hoping that it would have been enough.

- A first trial was done by means of the **Selective Search** algorithm proposed in the OpenCV library, which is a region proposal method designed to be fast with a very high recall. It is based on computing hierarchical grouping of similar regions based on color, texture, size and shape compatibility. The idea is to over-segment the image based on intensity of the pixels using a graph-based segmentation method. Unfortunately, the result was a complete failure, showing many false positives and not even getting close to some detection of the hands, not even after trying to merge the bounding boxes.

- Another trial was made trying to exploit the **Canny edge detection** algorithm. The idea was to keep only the person contour and then use other low-level techniques to keep only the hands. This method, however, revealed to be non-efficient due to the poor quality of the images and the many occlusions present in the photos.
- Finally, the last idea was to detect fingers using a line detection Hough transform. The idea arose out of the observation that the human fingers are way less deformable objects with respect to the hand. By detecting several straight "sort a" parallel lines we hoped to detect the fingers. As it turned out by having poor results on the edge detection this approach also failed. Within this approach with the (generalized) Hough transform we then tried to detect the fingertips. With a good approximation, human fingertip can be associated to half circles. For the same reason as above, we had to give up on this approach too early.

At this point it was clear that, for such a complicated task as object detection, low and middle-level processing approaches were not enough. Therefore, we tried to use some high-level computer vision algorithm and, given the abundance of data, we tried to pursue a Viola-Jones approach with a **cascade of classifiers**. It has been proved to be very effective for face detection, but hands can be considered deformable objects and can appear in so many different shapes, so we were very skeptical to get any results at all. As we will later show this was not the case, although it required some care in interpreting the detections.

2.2 Training of the classifier

The first approaches on the classifier training required a good understanding of the different parameters used in training. By means of the tools provided by OpenCV, i.e., *opencv_traincascade*, we tried many training sessions with different parameters involving, for example, the number of positive samples, negative samples, the training stages, the feature type etc.
This is an example of the parameters used on the *opencv_traincascade*.

```
numPos: 3500
numNeg: 7800
numStages: 25
precalcValBufSize[Mb] : 3500
precalcIdxBufSize[Mb] : 3500
acceptanceRatioBreakValue : 0.0001
stageType: BOOST
featureType: LBP
sampleWidth: 46
sampleHeight: 46
boostType: GAB
minHitRate: 0.999
maxFalseAlarmRate: 0.3
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
Number of unique features given windowSize [46,46] : 119025
```

The final result of the training then appeared this way.

```
Training until now has taken 0 days 3 hours 59 minutes 9 seconds.
=====  TRAINING 11-stage  =====
POS count : consumed   3500 : 3540
NEG count : acceptanceRatio    7800 : 7.22584e-05
The required acceptanceRatio for the model has been reached to avoid
overfitting of trainingdata. Branch training terminated.
```

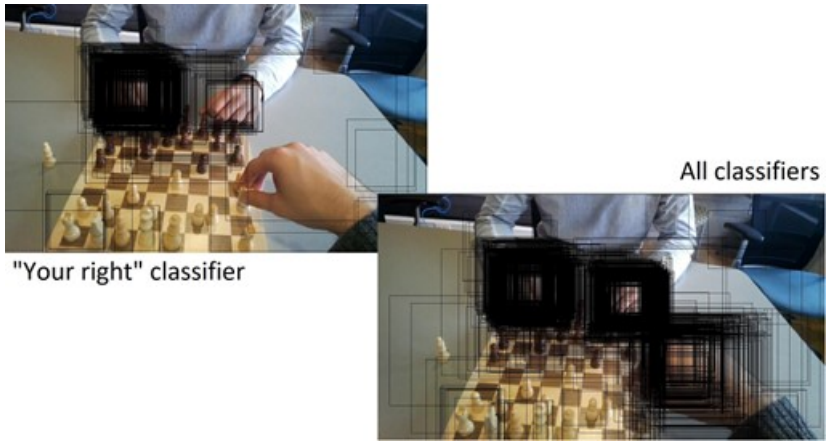
These parameters are very task dependent, but through a long phase of trial and error we found that the best outcomes in terms of prediction were mainly influenced by the number of positive images, the number of negative images, the sliding window (sample) size and, of course, the featureType. While the number of negatives, positives and sample size might be somewhat simple to work with (generally bigger numbers imply better results), it was not clear at all what feature type to use with (HAAR vs LBP). Not even after going through the OpenCV documentation it was explicit which one was the best. As it turned out after different trials, LBP features were way faster to perform the training, and gave overall better results in terms of hand vs non hand predictions. Another counter-intuitive result that we found relies on the idea of using segmented hands, extracted from the ground truth provided in the dataset, as positives in the training phase.

We expected our classifier to be more precise if trained on the segmented positive images (black background) rather than the one with its background, but it was quite the opposite. This aspect could be worthy of further and deeper investigation.

In the early development stages, we were using a single classifier for everything. This approach was later dropped in favor of 5 different classifiers, each one trained in the 5 listed image groups: {"my left", "my right", "your left", "your right", "hands over face"}. This way the classifiers became more specific and were able to tell exactly how every classifier was working and the ability to not just find hands in the image, but also to label them in one of the 5 above groups.

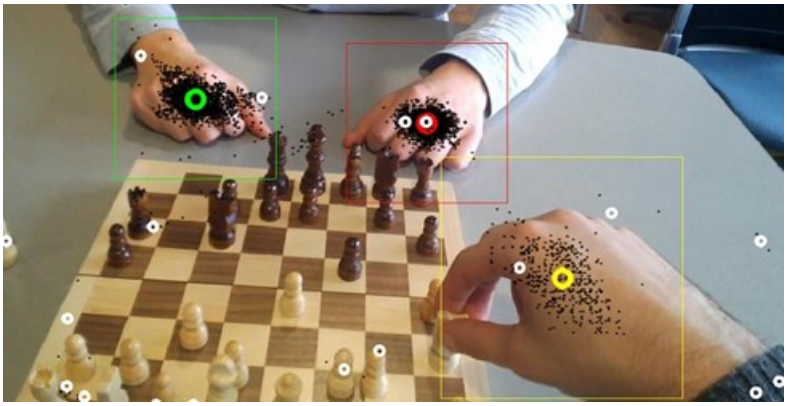
2.3 Detection process

Hand size in an image can vary considerably and it can easily vary from occupying a small portion of it to being the principal element. We had to abandon the idea of detecting hands as objects, while instead focusing on a somewhat probabilistic model to say that in a particular area of the image there is probably a hand. After running the detection, we faced the following scenario:



So, we basically did not have a single rectangle containing a hand but rather many of them, of many different sizes. In this situation, we obviously could not rely on the biggest bounding boxes, because there are several of them also in the non-hand area. Therefore, the idea was to discard the rectangles entirely and consider only the distribution of their centres.

The black dots in the image represent detected features (the centers of the rectangles). In the single classifier case, we faced the problem of clustering the black dots, which was certainly feasible, but led to more complex clustering problems. First, by using a **k-means algorithm** we needed to know in advance the number of clusters, and there were many false positives in this scenario. We then came up with the idea of using the **mean shift algorithm** to "climb" to the peaks of the distribution of the centres and to not have a fixed number of clusters, so that the system would have been more robust to images with different numbers of hands. This approach, combined with the use of 5 different classifiers, worked very well. As shown in the above example of the "your right" classifier, the task became much simpler. It is



still using the mean shift algorithm to find the "centre" of the clusters, but in this case, we classify as a hand the cluster with more points in it. The white circles in the image represent other excluded clusters.

2.4 Creation of the bounding boxes

At this point, prior to advance with the segmentation phase, we had to determine the bounding boxes of the detected hands. The only information we had out of the detection phase was the distribution of the black dots, but nothing regarding their size. The idea was to construct the bounding boxes centered in the center of the cluster and find a suitable measure for their size. The idea behind this was derived from the following observation. The denser the dots are in the cluster, the smaller the hand is. This is because all the features are concentrated in a smaller area. On the contrary, the sparser the points of the cluster, the bigger the hand. Therefore, we used the average distance from the centre for all the points in the cluster in order to determine how big the bounding box should be. This had the drawback of creating only square bounding boxes, because we had information about the position of the hand, but not on its "dominant direction", therefore it was impossible to use rectangular bounding boxes, but we rather had to stick with squared ones. This lowered the final results even though, at least visually, the detection was successful.

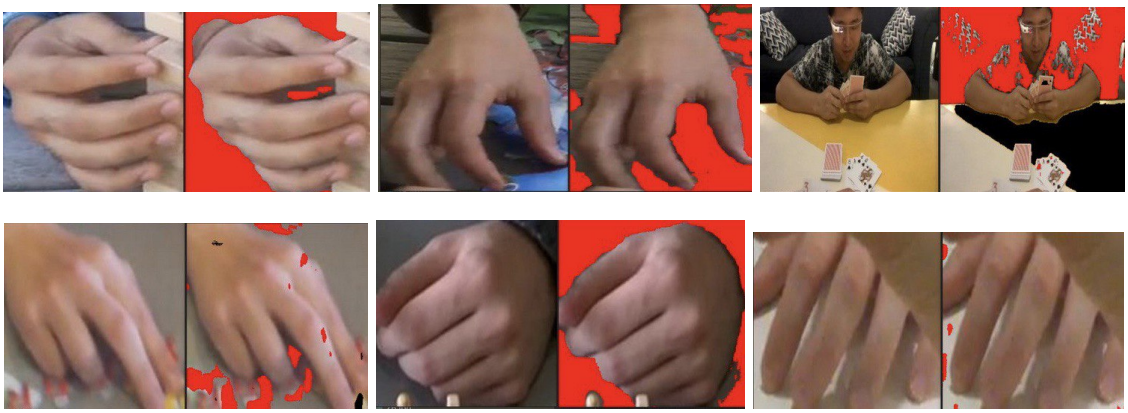
3 Hand Segmentation

Starting from the detected hands in the previous step, it was then necessary to develop an algorithm able to segment them from the background.

- Our first thought went to the idea of **filtering the image using colour spaces**, in order to detect the skin and separate it from the rest of the scene. Many colour spaces were tried, among which HSV, LUV, YCrCb. In fact, by using the thresholding function *cv::inRange*, and playing with the lower and upper bound values in the different colour spaces, the idea was to create a binary mask that had white pixels where the skin was, and black pixels elsewhere. Here there are some examples of images filtered in the **HSV space**, and it is possible to see that some images were very well segmented, but others were completely out of the goal. It was clear that it was not an approach that could have been applied globally, also because the lower and upper values had to be adjusted manually for each single image.

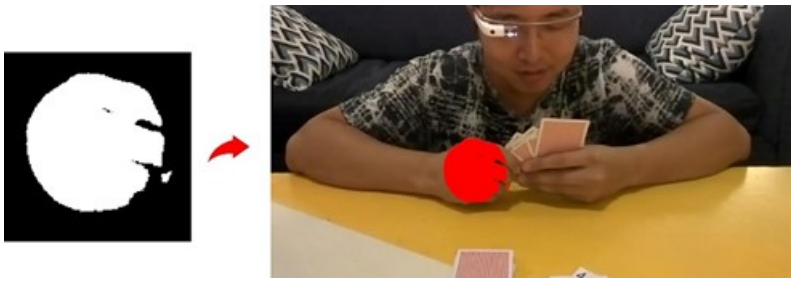
- Another trial was done in the **LUV space**. In this context, we obtained three different LUV values

that represent the skin; these values were the result of a k-means algorithm (3 clusters) run over the whole dataset of the segmented hands. This method relies on these three clusters and measures the absolute distance (Manhattan distance) between each pixel and 3 LUV values. The following step would be, similarly to before, to set two threshold values based on the 3 clusters in order to classify each pixel. Two regions are defined by the thresholds, in particular: above the higher (red) and below the lower (black). The idea was to apply this method prior to the actual segmentation method in order to drive it more accurately. However, it works correctly on average, but leads to a hard misclassification in some cases. It was therefore not convenient to use it for the final program, however in our opinion it was worth it to mention it. In the following we can see some results of this thresholding method in the LUV space.



As it turned out, a simpler, more naive approach to this problem resulted in overall best results. The idea was to do a **segmentation by thresholding** using Otsu's method. To refine and help the thresholding process, two filters were applied to the hand images in advance. First, we performed a **bilateral filter** in order to smooth the area and after that we perform a **"vignette" filter**.

The latter was created by darkening the pixels starting from the centre and then in a radial path we darken the pixels. This resulted in a darker area near the borders of the bounding box, where it's less probable that hand pixels might be, and a lighter central area, where we're almost sure a hand is. Finally we applied the thresholding and coloured the segmented area accordingly.



4 Detection and Segmentation System

4.1 Structure of the program

The implementation part of this project is substantially divided into two main parts:

- Detection/segmentation
- Evaluation of the system

Let us briefly describe what the program does and how to run it by terminal.
In the file "*Detection_Segmentation/handDetection.cpp*" we find all the implementation of the two parts described before. First, all the 5 trained classifiers are loaded to be used for the detection. Afterwards, the images of the test set are displayed one at a time, and to go from one to the next it is just sufficient to press any button (e.g., space bar). Once the image is loaded and displayed, together with it we have a whole set of other windows showing on the screen, that are:

- "*Detections*": the window containing the image with the detected bounding boxes;
- "*Colours*": the window in which the original image is displayed with the segmented hands, with different colours for each hand;
- "*BW*": the window displaying the binary mask containing the whole segmented image;
- For the i-th detected hand:
  - "*Rect*: i": displays the content of the bounding box;
  - "*Vignette* : i": displays the effect of the vignette filter;
  - "*Threshold* : i": displays the Otsu thresholded image after the vignette filter.

With each image, the user has also at disposal some trackbars to play with, that allow to change some detection parameters, such as the window size of the classifier features, or the kernel window size of the mean shift algorithm etc. By default, they are set to the combination of parameters that gave us the best results.
The second part, instead, ,under the "*Evaluation*" folder, is based on the results of the previous one, and we implemented and used it to evaluate the system. It allows to display the Intersection over Union values (see the Results section) for each bounding box and also shows the average values for each image and through the whole test dataset. Moreover, it computes and displays the pixel accuracy.
To change between one or the other program is sufficient to uncomment the relative *add_executable* string in the *CMakeLists.txt*.
For example, in the following setting the evaluation part will be executed:

```
#add_executable(HandDetection Detection_Segmentation/HandDetection.cpp
    Detection_Segmentation/meanshift/MeanShift.cpp)

add_executable(HandDetection Evaluation/main.cpp
    Evaluation/include/evaluate.h Evaluation/include/evalute.cpp)
```

To run the program, it is sufficient to open a command window inside the main folder and type:

```
$ cmake .; make ;./HandDetection
```

So that it builds, compiles, and runs everything. If in the evaluation part one wants to display the windows with the IoU values for each hand, it is just necessary to add an argument, while for saving the displayed images the second argument has to be the string "*save*". For example, the following instruction will both display and save the images:

```
$ ./HandDetection plot save
```

Let us also explain briefly the structure of the project folder.

- The *CMakeLists.txt* file is in the main directory, hence that is the location from which any command line shall be executed.
- The folder *Detection_Segmentation* contains the source files to execute the above explained system. In the empty folders *our_detection* and *our_colored* the processed images are saved with the parameters chosen by the user.
- The folder *Evaluation* is responsible of the evaluation part described before. Inside the *images* subfolder one can find all the IoU images and binary masks, and if the user wants to save them while running the program, those will end up in the *output_detection* folder.
- The folder *Results_Final* contains all the results obtained, from the bounding boxes coordinates to the binary masks to the images that are also reported in the appendix.

5 Results

5.1 Intersection over Union for detection

In this section the results of the detection system are reported. As a metric of evaluation, it was requested to compute the Intersection over Union (IoU) between our detection and the ground truth. In the following table are reported the single IoUs between each bounding box, the average IoU per image, and the average IoU over the whole test set. Moreover, a couple of output images taken from the dataset are shown, while the rest of them can be found in the appendix. Since usually a detection can be considered good when the IoU is higher than 0.5, such values are highlighted in green. In yellow we can find $0.4 \leq \text{IoU} \leq 0.4999$, while in red are highlighted the IoUs smaller than 0.3999, that therefore cannot be considered a good output.

Regarding the first table, *myLeft* and *myRight* are the hands of the person of the "POV", the one taking the photo, while *yourLeft* and *yourRight* are those of the person in front, on the other side of the table. In general, the *Xs* are placed where a certain hand is not present in the image.

# IMAGE	MY LEFT	MY RIGHT	YOUR LEFT	YOUR RIGHT	AVERAGE IoU
1	X	X	0.581827	0.637869	0.609848
2	X	X	0.608869	0.688417	0.648643
3	X	X	0.720655	0.411524	0.56609
4	X	X	0.399286	0.458182	0.428734
5	X	X	0.387515	0.648763	0.518139
6	X	0.528317	0.42796	0.59229	0.516189
7	X	0.553845	0.510507	0.529442	0.531265
8	X	X	0.593992	0.632856	0.613424
9	X	X	0.41949	0	0.209745
10	X	X	0.57346	0.58626	0.57986
11	X	0.627946	0.529595	0.311032	0.489524
12	0.521257	0.643666	0.54474	0.41112	0.530196
13	0	X	0.868588	0.680501	0.516363
14	0	0.504111	0.648004	0.752806	0.47623
15	X	0.403417	0.527363	0.472775	0.467852
16	0	0.197856	0	0.529595	0.049464
17	X	0.49759	0.532585	0.82889	0.619688
18	0.535447	0.488131	0	0.477417	0.375249
19	X	0.0436145	0.411797	0.758505	0.404639
20	X	0.636668	0.370987	0	0.335885

IoU over the EgoHands test set

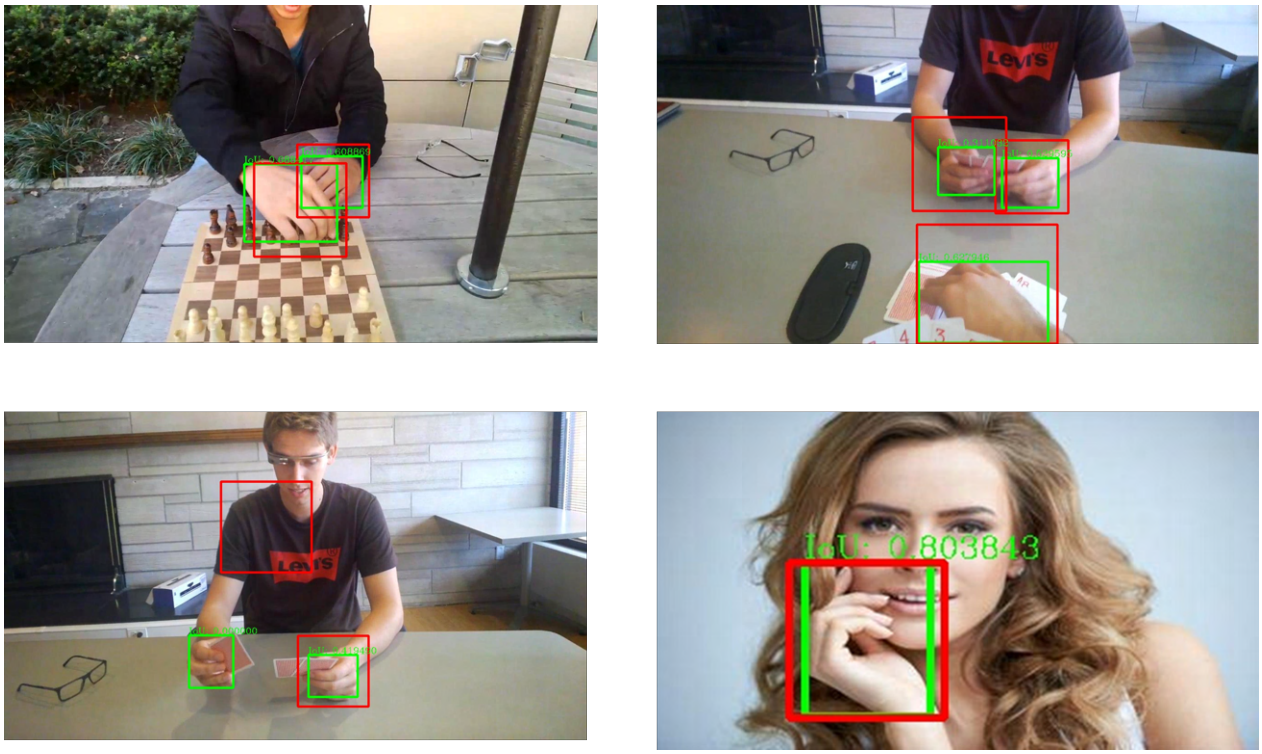
# IMAGE	LEFT	RIGHT	AVERAGE IoU
21	0.403161	X	0.403161
22	0	0.26162	0.13081
23	X	0.803843	0.803843
24	X	0.394963	0.394963
25	0.380014	0	0.190007
26	0.441643	X	0.441643
27	0	0.548966	0.274483
28	X	0.295969	0.295969
29	X	0.24012	0.24012
30	X	0.444178	0.444178

IoU over the HandsOverFace test set

The average IoU over the whole dataset is **0.436873**. It is also worth mentioning that the whole test set is extracted from two different datasets (Image 1-20, Image 21-30), therefore let us also report the two separate average IoUs:

- EgoHands: **0.47435135**
- HandsOverFace: **0.361917**

The obtained results are not satisfactory from a mere numerical point of view, because the minimum value of 0.5 is not reached in the average computation. However, let us consider that the bounding boxes in output from our system are squared, which is very frequently in contrast with the ground truth boxes that, instead, accurately surround the hand. Therefore, from an absolute point of view, considering also the "visual" output that can be observed in the following images and the ones in the appendix, the overall result is not that bad.



Images 02, 09, 11, 23

5.2 Pixel accuracy for segmentation

In this section are reported the results for the segmentation part. The metric used here, instead, is the pixel accuracy that, in a binary segmentation, can be considered as the following ratio:

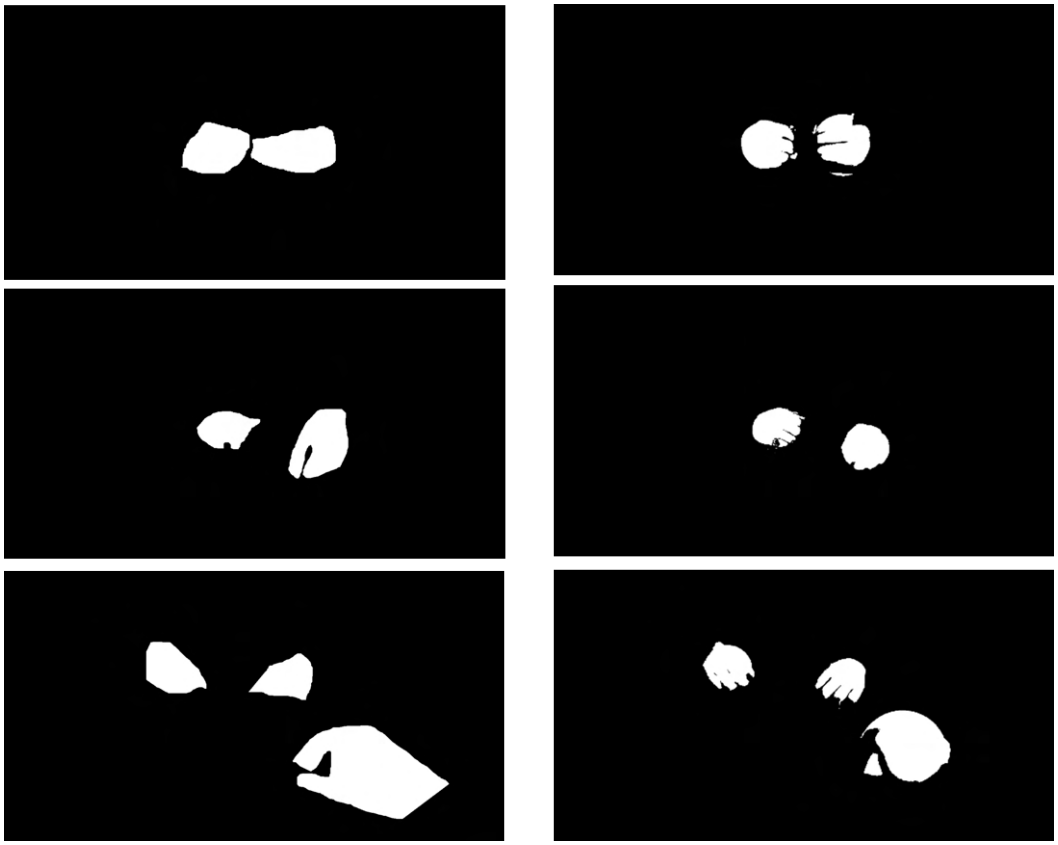
$$\frac{TN + TP}{TN + FN + TP + FP}$$

In our case, the true positives are the hand pixels segmented as hand, and the true negatives are the non-hand pixels segmented as non-hand. The denominator is just given by the total number of pixels.

# IMAGE	PIXEL ACCURACY
1	0.981275
2	0.988771
3	0.986745
4	0.99175
5	0.983974
6	0.956849
7	0.936236
8	0.980352
9	0.965357
10	0.988633
11	0.96435
12	0.88745
13	0.957818
14	0.943609
15	0.975149
16	0.873869
17	0.938449
18	0.840859
19	0.921
20	0.973588
21	0.93058
22	0.949086
23	0.972548
24	0.958587
25	0.84041
26	0.884862
27	0.950593
28	0.922502
29	0.907769
30	0.977358

Pixel accuracy over the test set

The average pixel accuracy over the whole test set is **0.944346**. Also in this case it is opportune to give an interpretation of the obtained results. From the pixel accuracy values, one may say that it is almost perfect (very close to 100%). What happens is that the pixels of the background are, on average, very much more than the ones belonging to the hands, since the hand is a part that usually does not fill the image so much. This may cause the ratio to be close to 1 even if the segmentation is not visually very precise, because the larger region is given by the background. Let us report a couple of examples of comparisons between ground truth masks and our segmented masks (Image 01, 03, 07). As always, in the appendix every other image can be found.



Images 01, 03, 07 (Ground truth on the left and segmentation output on the right)

6 Conclusion

This report explains the reasoning, approaches and techniques implemented by our group to solve the detection and segmentation task. Many methods were tried with poor results, before reaching an acceptable level. Since the task is a high-level processing problem, it was difficult to be tackled by means of low and mid-level algorithms, however some of them were not as bad as expected. The detection and segmentation system could definitely be improved, for example by using more high-quality images for the training of the classifier, or developing a more complex system to find also the dominant direction of the hand in order to draw rectangles following better the shape of the hand, thus increasing the IoU values.

6.1 Contributions to the project

The amount of work done by the three of us was equally distributed, we tried to cooperate and share common ideas as much as possible since the beginning. At the first stages of the development of the project we worked together all along, in order to understand how to deal with the task. We tried to work as much as possible in team, in order to have different points of view on the same argument and exchanging opinions, but if we have to define the individual parts this is how, more or less, each component gave its contribution:

- Ralton mostly took care of the classifier training and came up with the idea of the vignette filter for the segmentation;
- Andrea worked on the segmentation task since the beginning, trying many different approaches mainly regarding color skin detection and edge detection filtering;
- Stefano worked also on the segmentation, and then mostly took care of the evaluation metrics, the testing on the Taliercio virtual machine and the finalization of the report.

After confronting each other, we can say that each one of us worked between 60 and 70 hours on the project, including both individual hours and group working hours.

7 Appendix

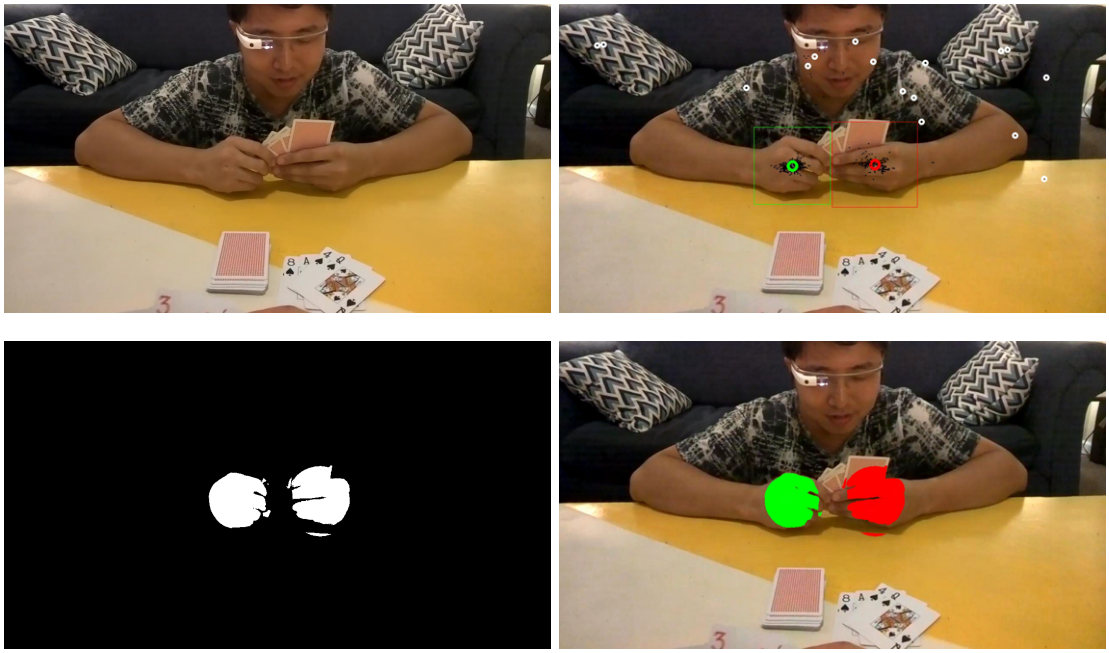


Image 01

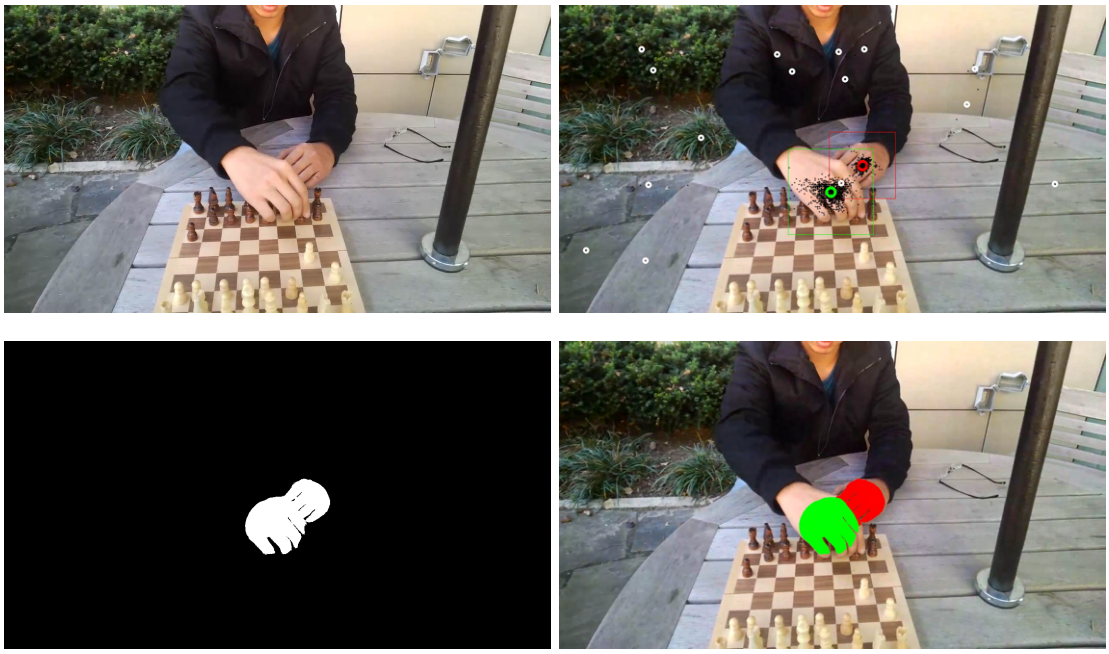


Image 02

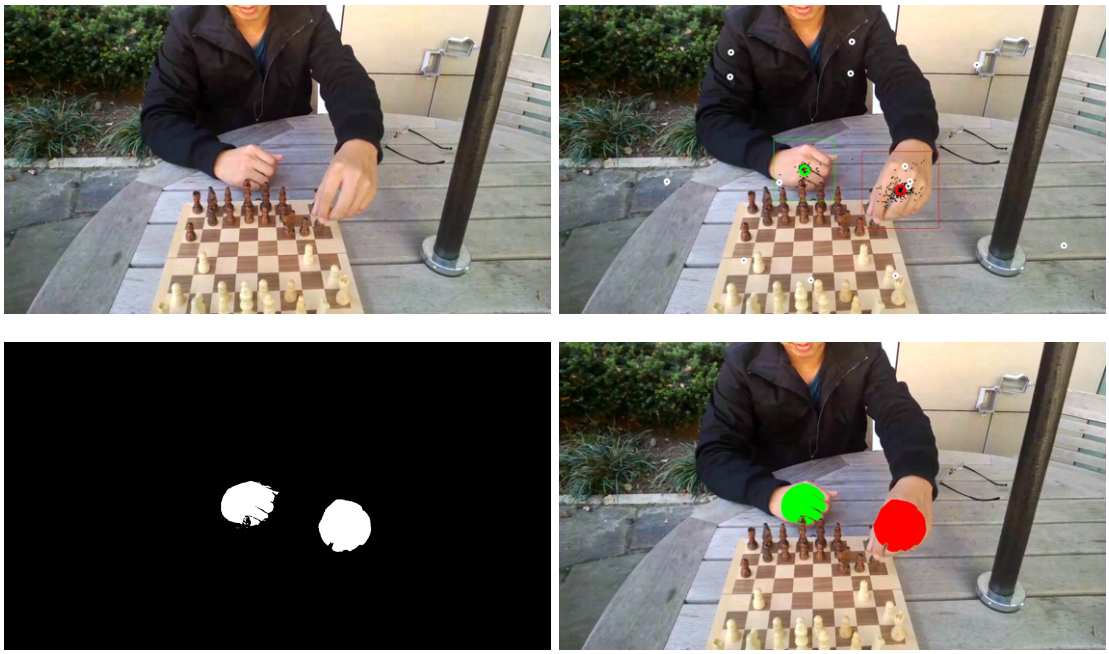


Image 03

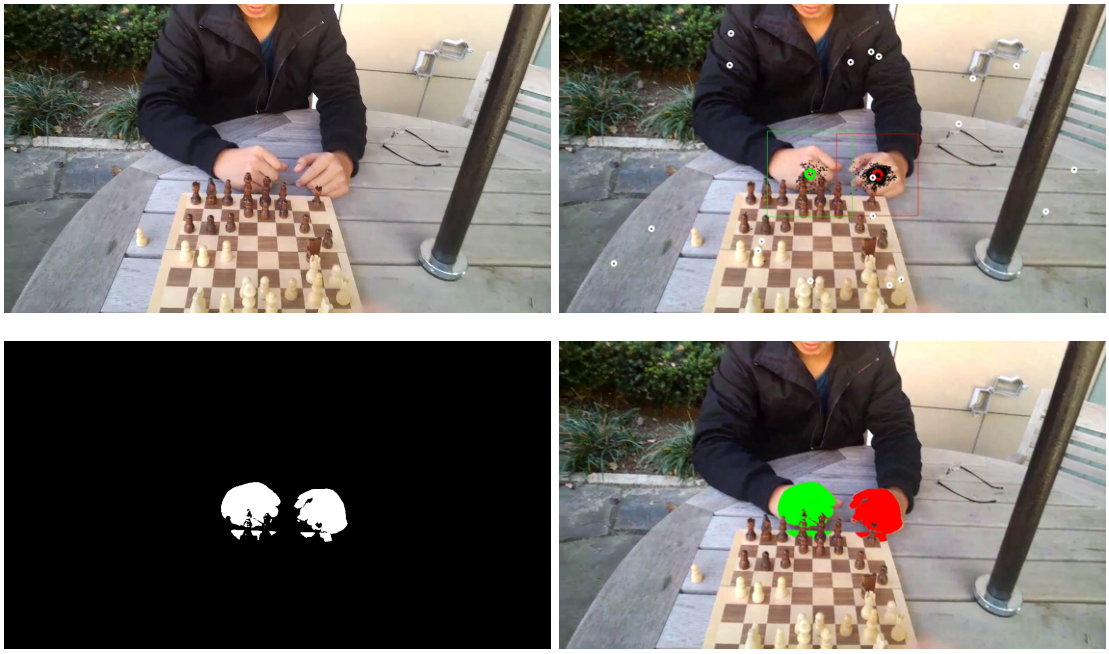


Image 04

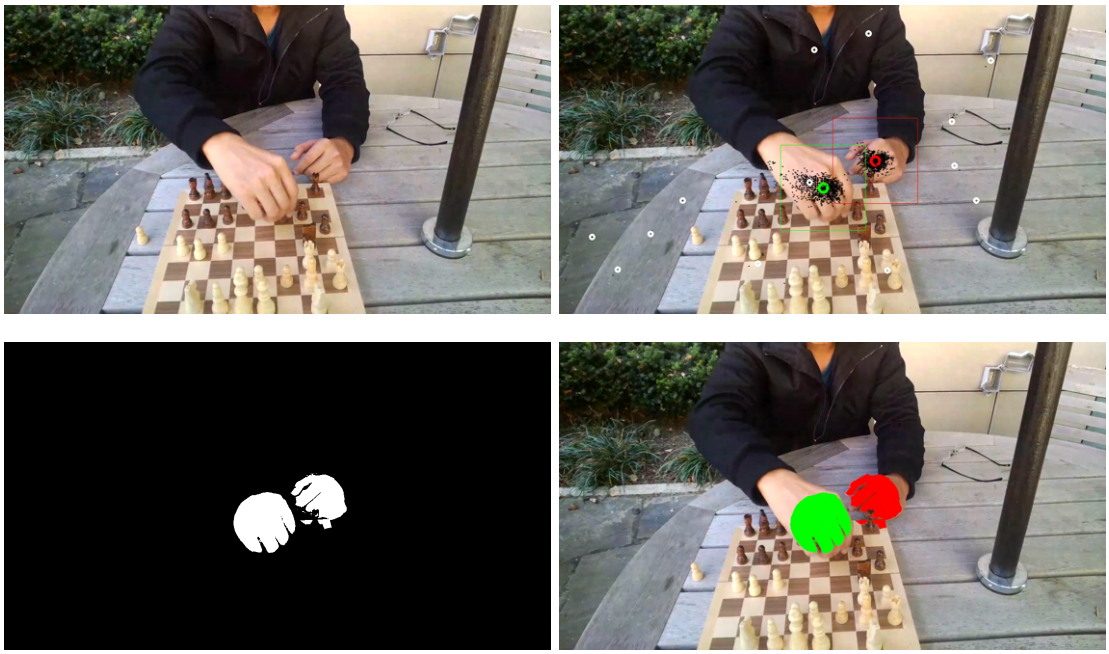


Image 05

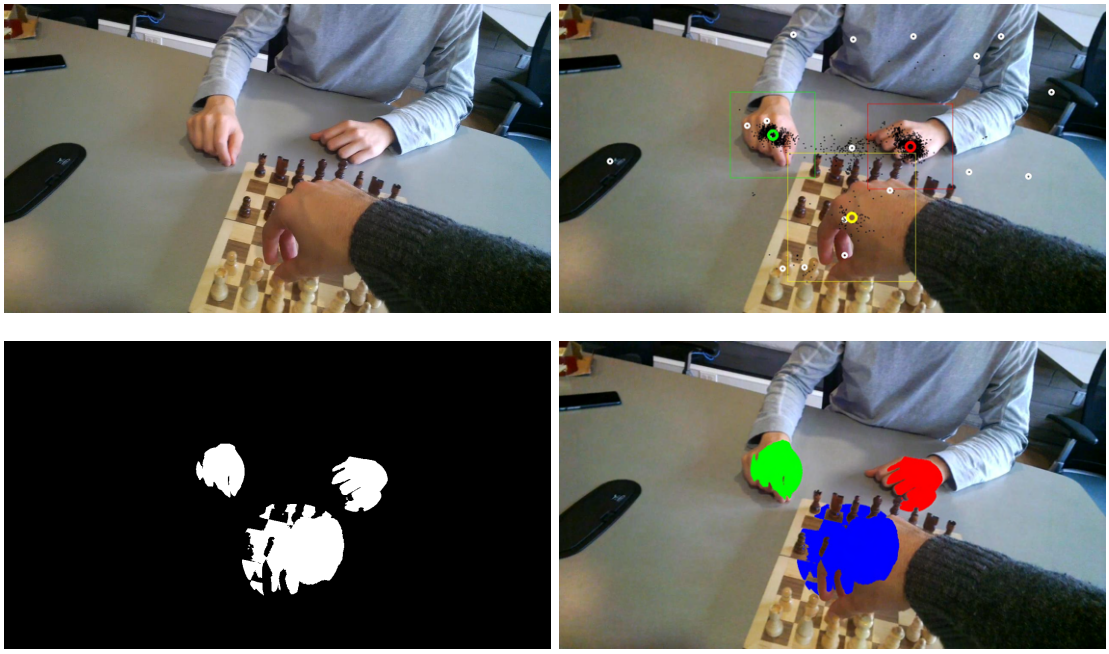


Image 06

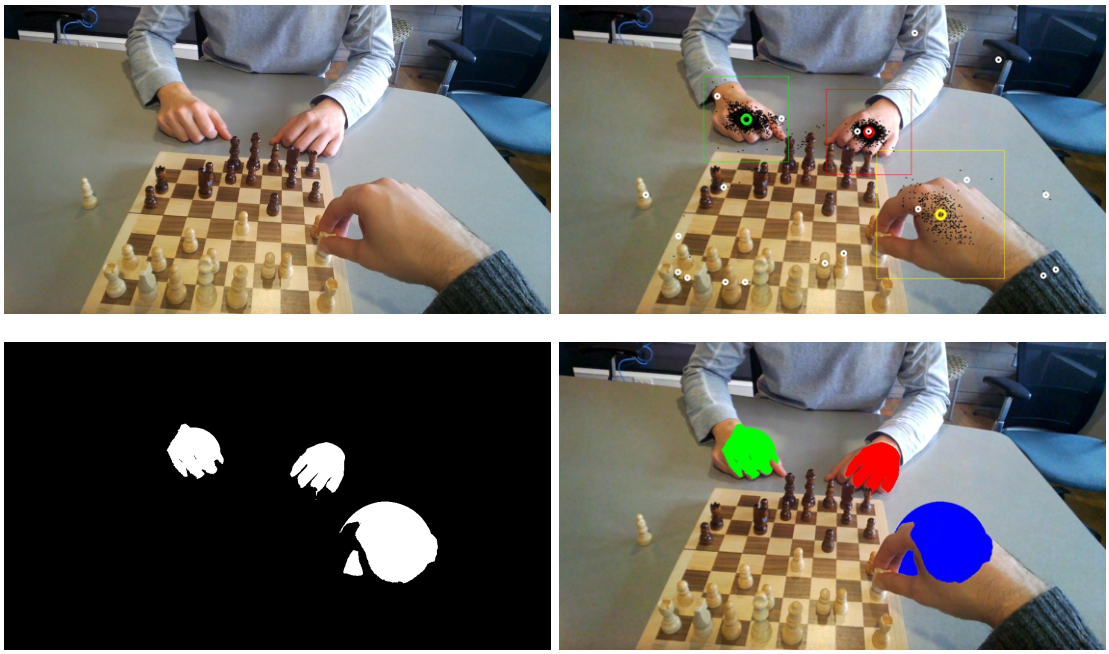


Image 07

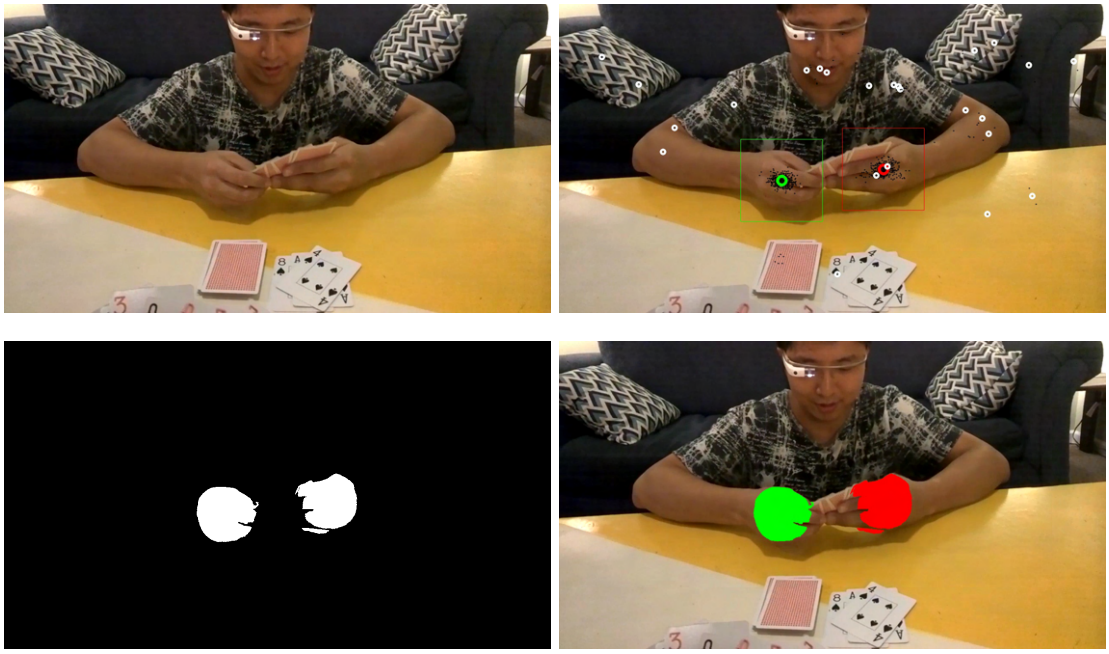


Image 08

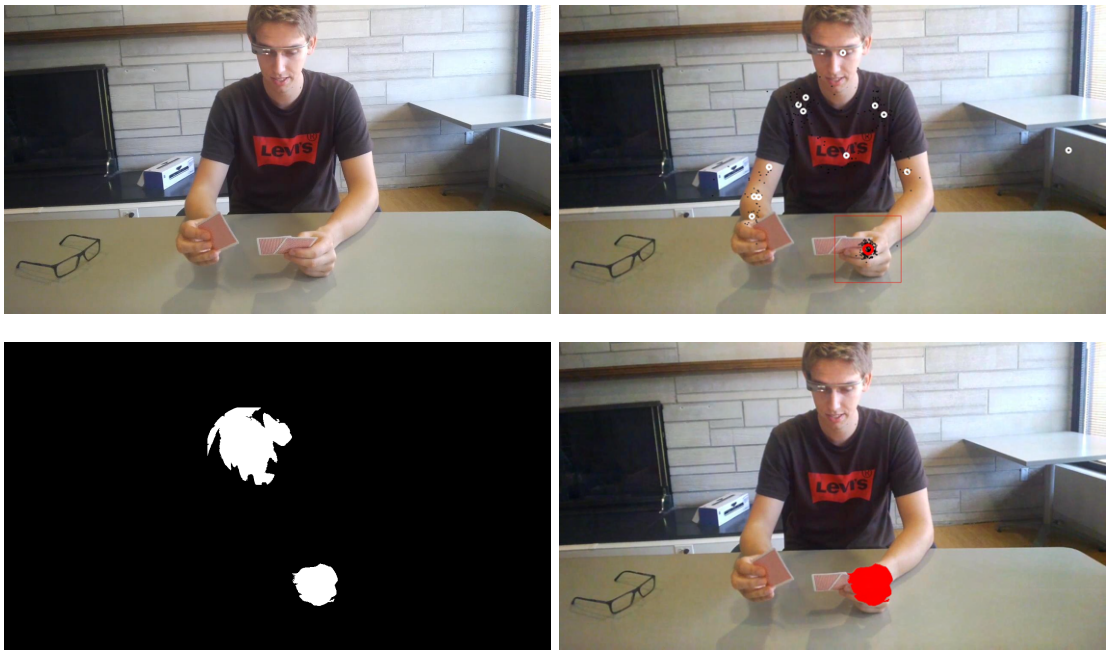


Image 09

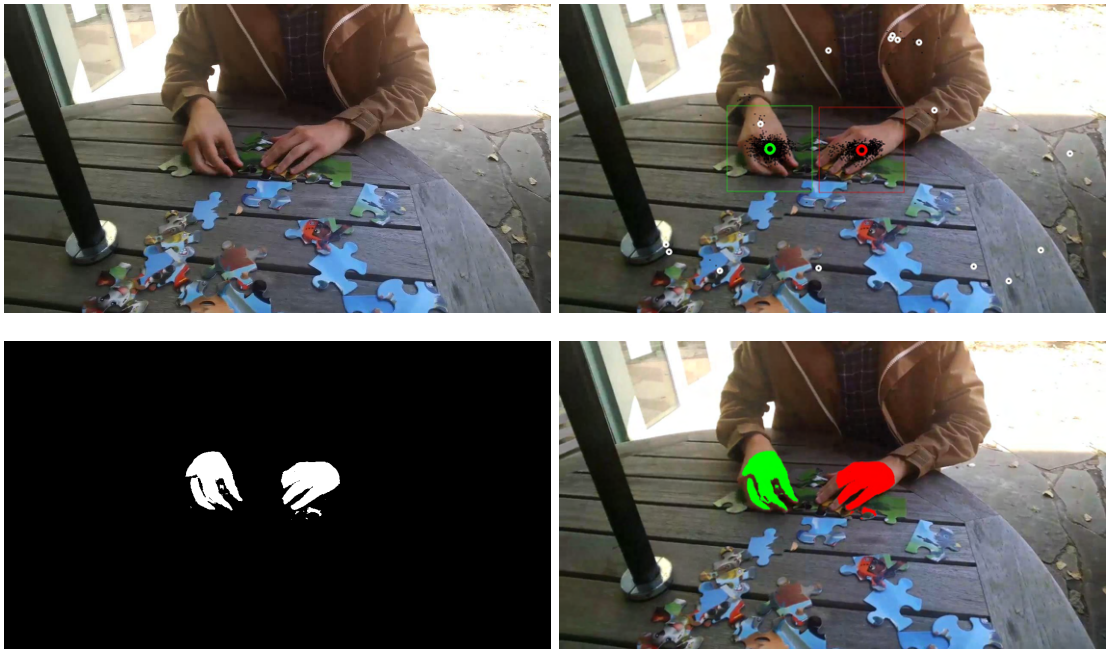


Image 10

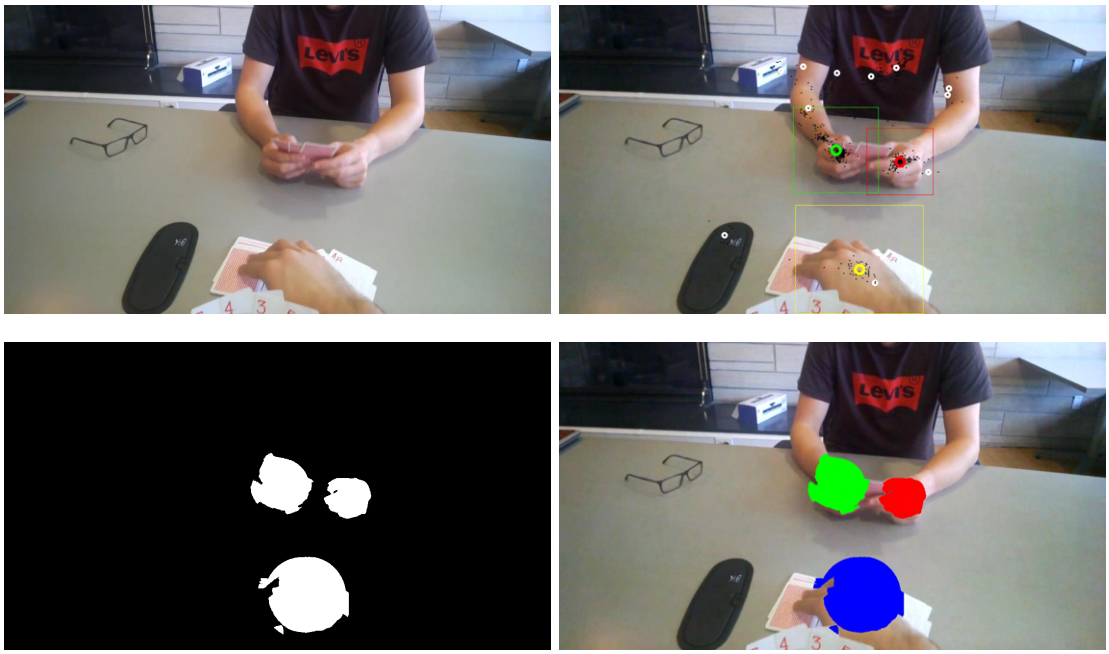


Image 11

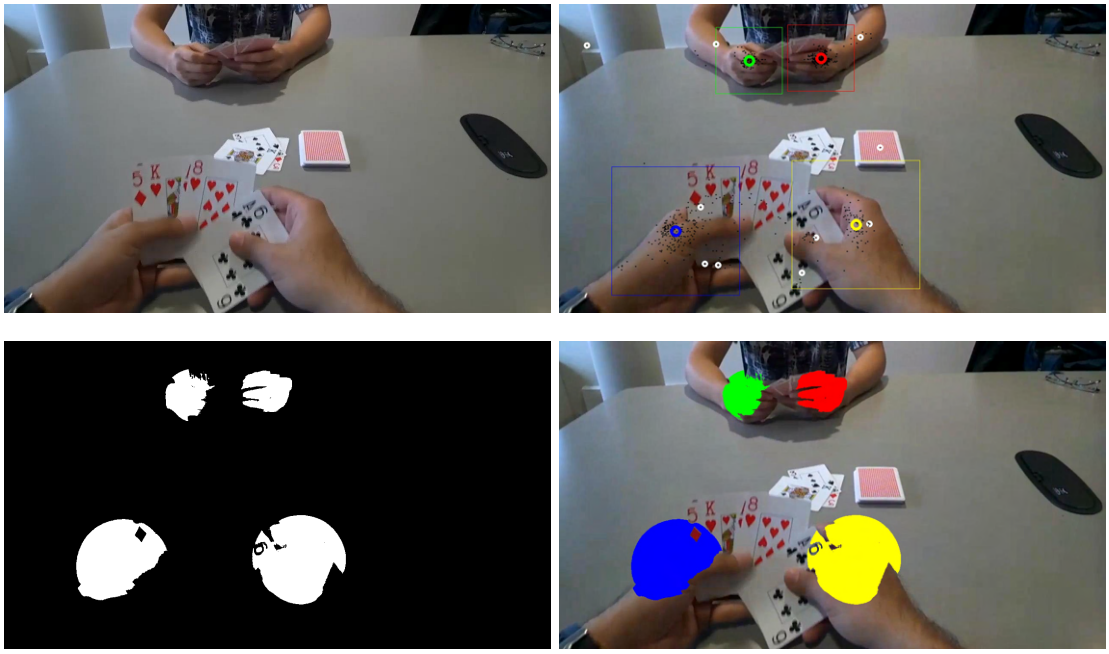


Image 12

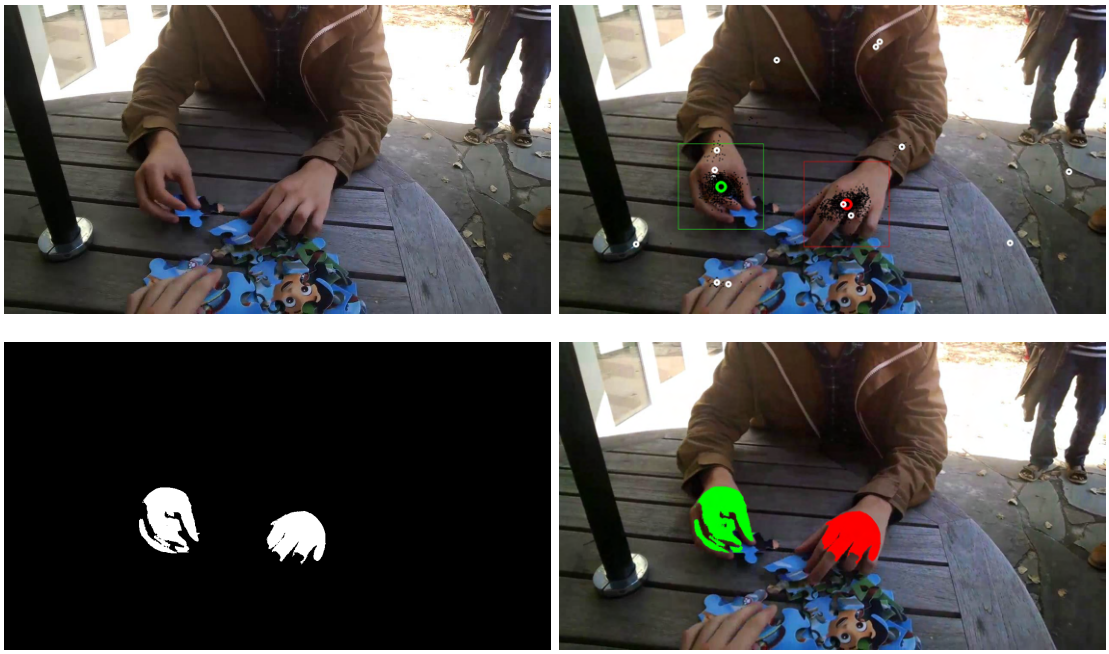


Image 13

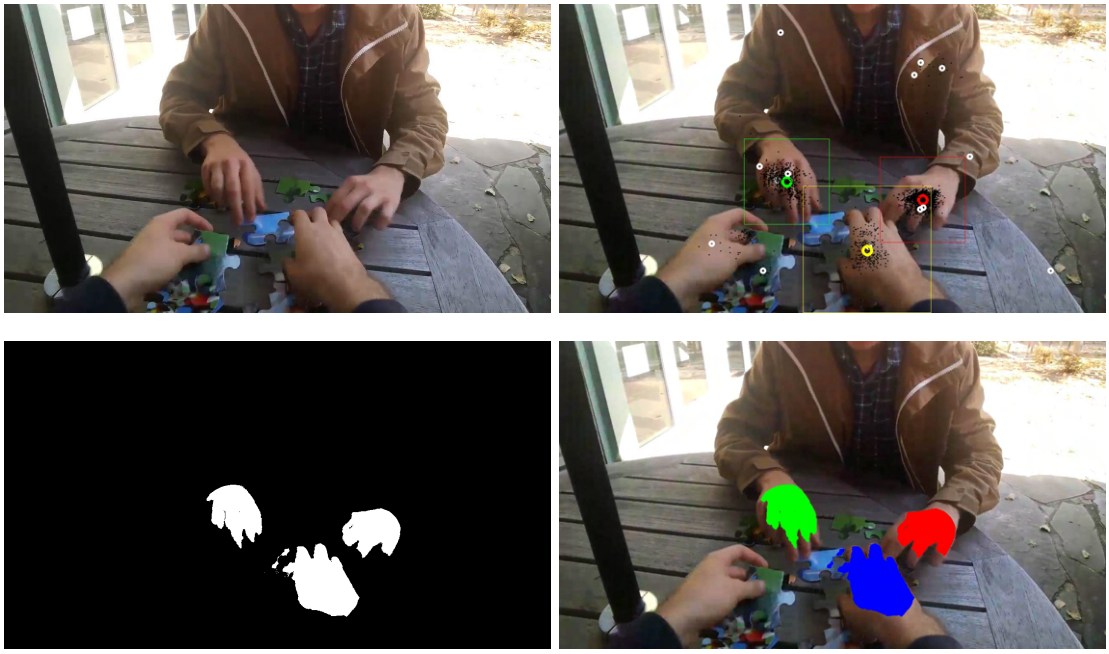


Image 14

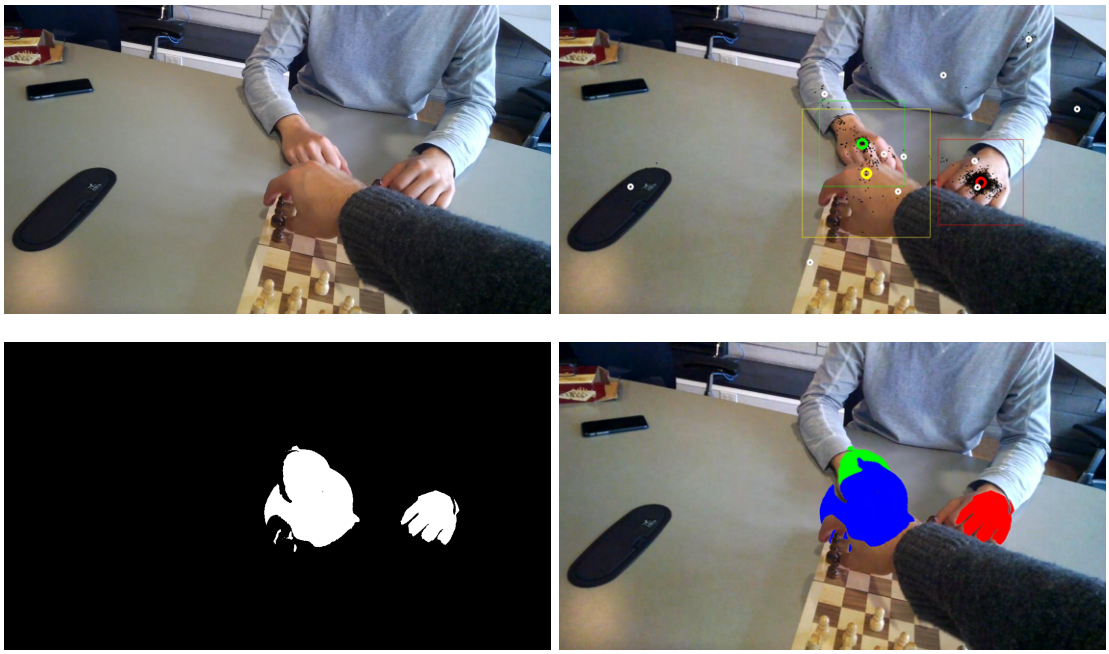


Image 15

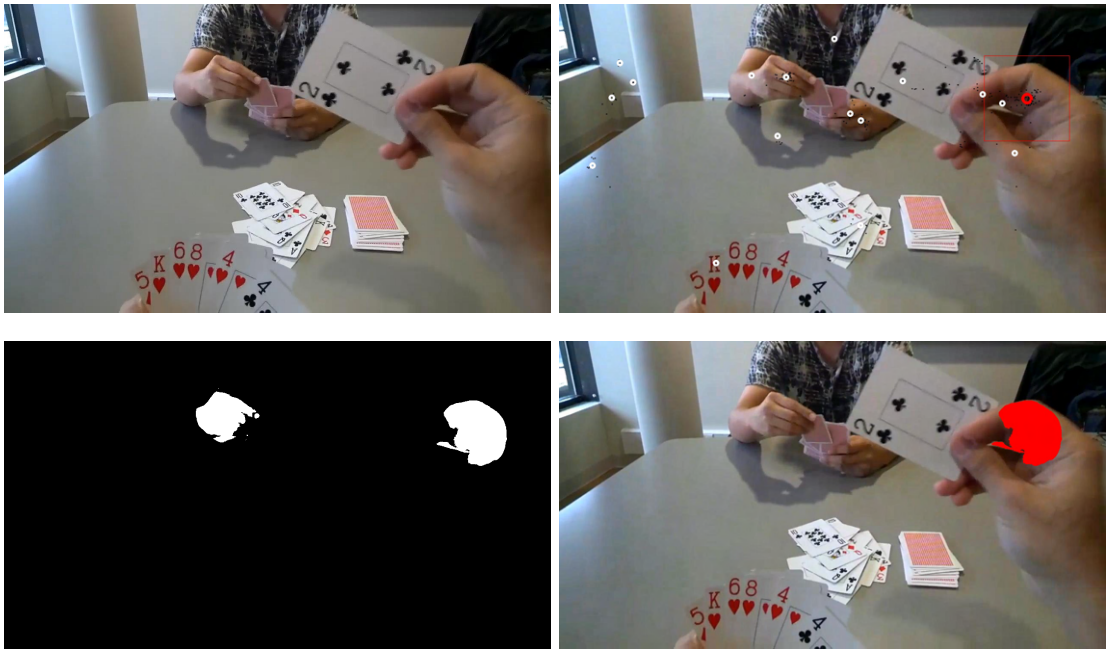


Image 16

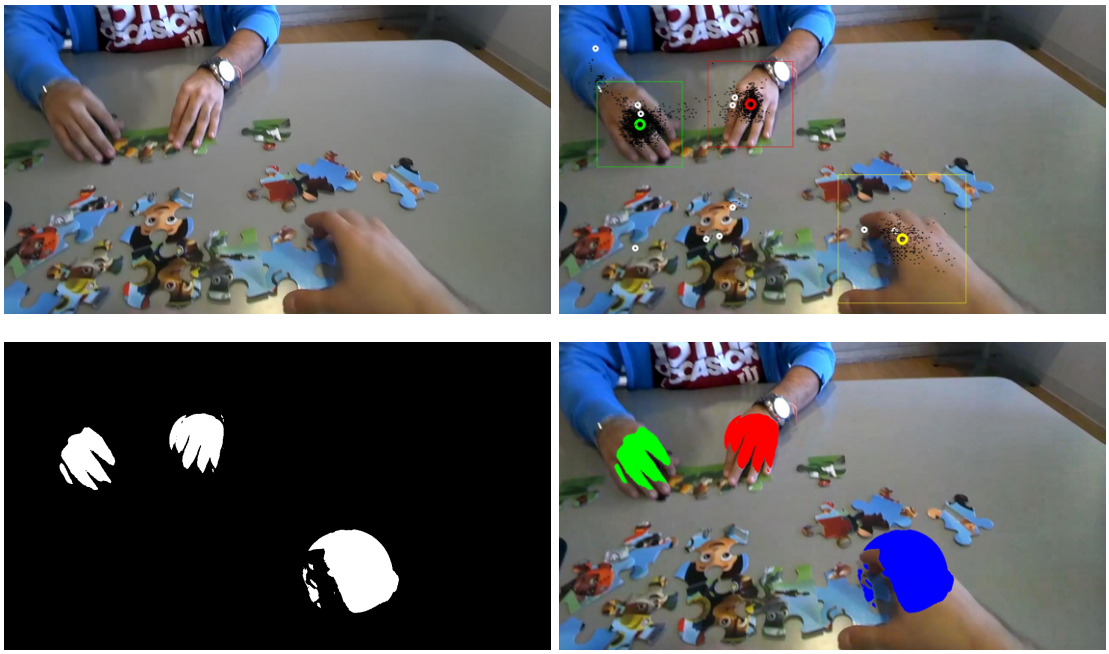


Image 17

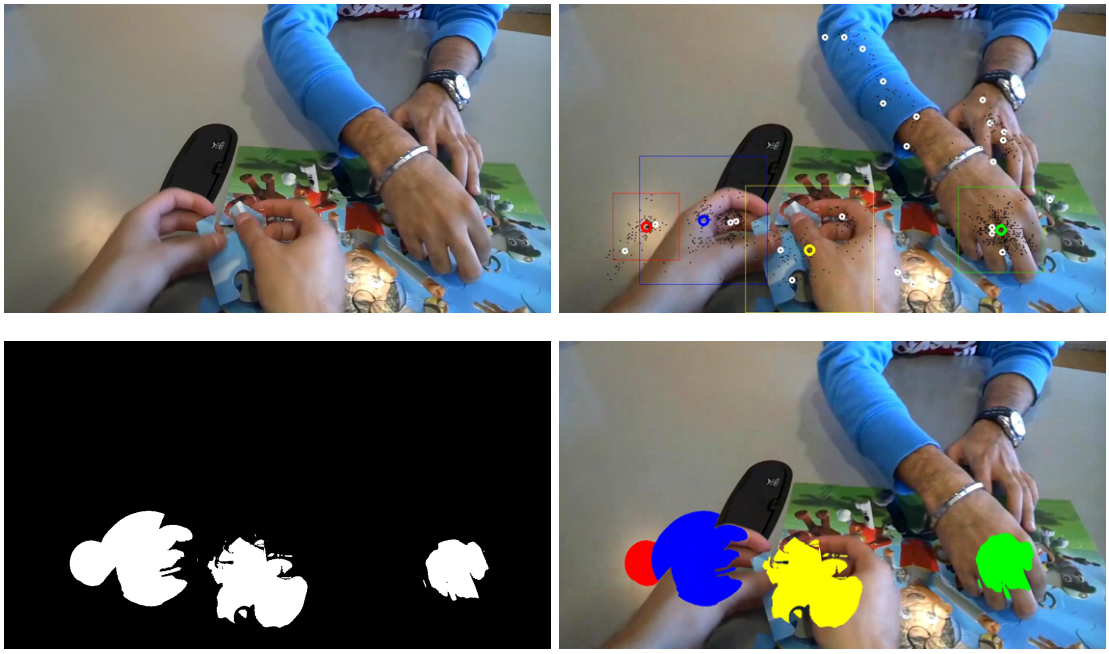


Image 18

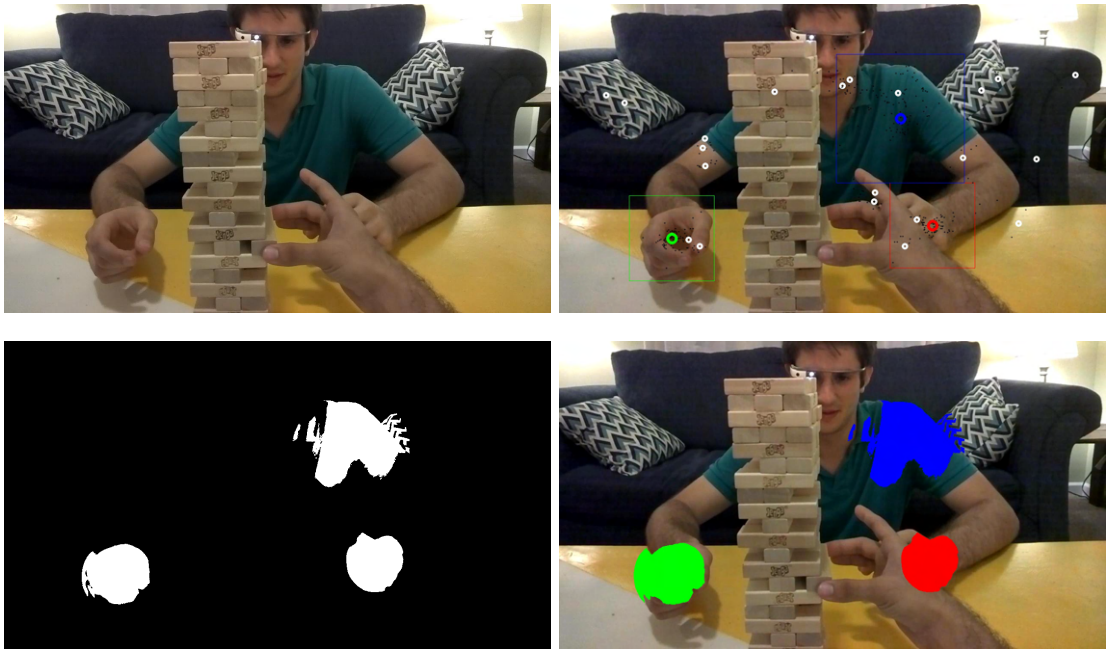


Image 19

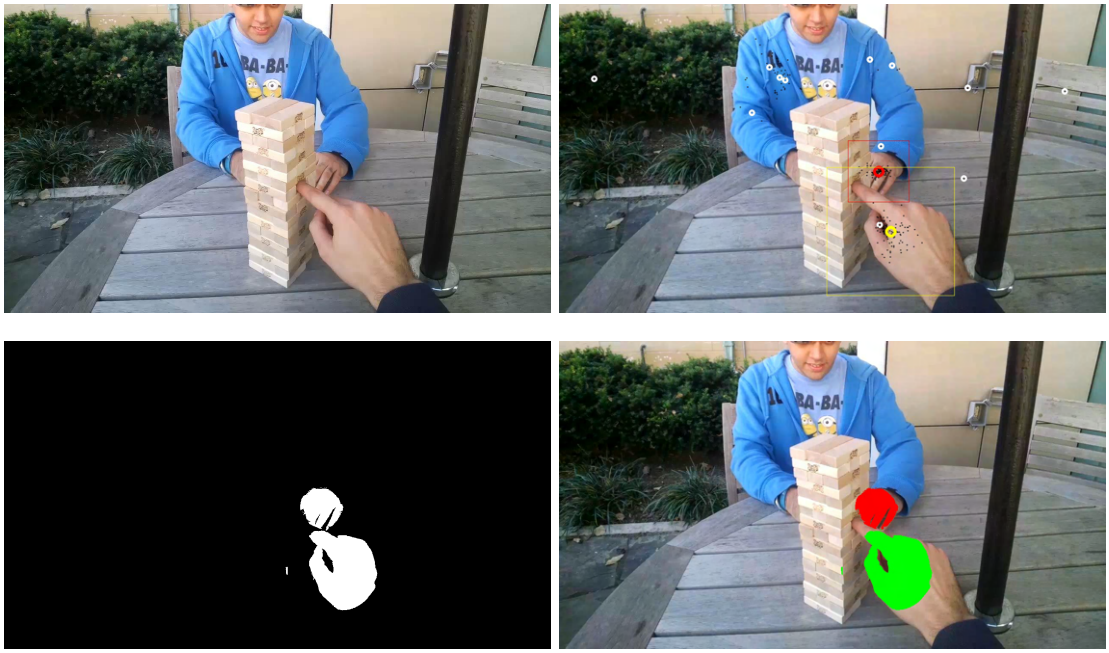


Image 20

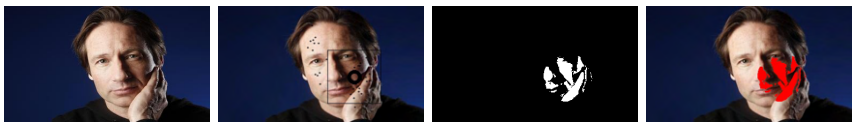


Image 21

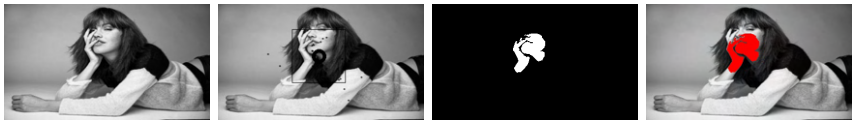


Image 22

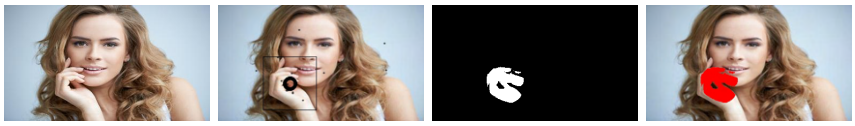


Image 23

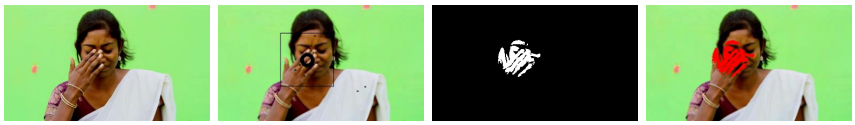


Image 24

Image 25

Image 26

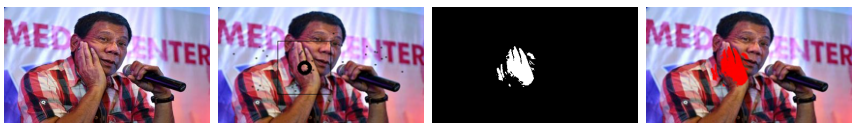


Image 27

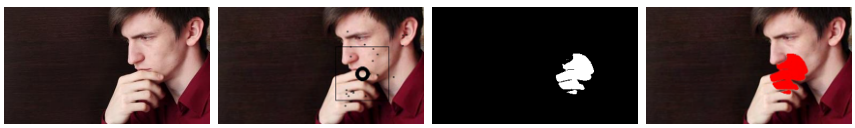


Image 28

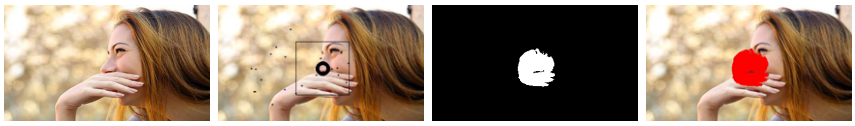


Image 29

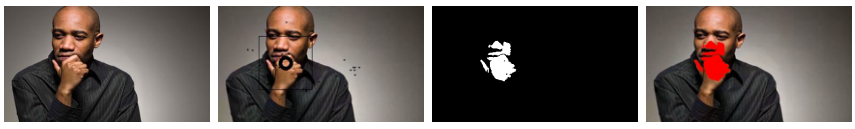


Image 30