An Augmented Reality Application with Hand Detection and Gesture Recognition

Alberto Zancanaro† - ID: 1211199

***Index Terms*—Saturn V, Augmented Reality, Hand Detection, Gesture Recognition, Vuforia, PyTorch, OpenCV, C#.**

I. Introduction

In this project that I developed for the course of *3D augmented Reality* I create an application that show a 3D model of a Saturn V and some related objects. In this report I will briefly introduce the project, its development history and what it shown. After that I will discuss deeply the various part of the project, especially the various scripts. In the end I will the discuss what I learn during this work and what are the possible future improvements.

A. Development history

The initial prototype was developed as test to learn how to use Unity and Vuforia. It works but the results were limited; basically the application only show the model in an augmented reality environment but all the iterations were made through mouse and keyboard input. So it was more a normal 3D application with a background capture in real time.

After some research I manage to better understand how Unity works and how other application can interact with it[1]. Following a tutorial I managed to create a python script that interact with Unity and track the hand position in the left side of the screen and check if the hand in the right side of the screen is open or close. Through this script I manage to implement some simple gesture command inside the Unity application.

This approach work but had several limitations, especially the division of the screen and the deterministic algorithms on which it is based. To improve the performance of the hand tracking and the finger detection I decide to implement them through neural networks. More specifically I use a Fast-RCNN ([2], [3]), a particular type of RCNN (Region with Convolutional Neural Network [4]) to track the hands in the picture. Once that the hand are found I feed them into a simple CNN to evaluate the number of fingers raised.

B. The Saturn V

The Saturn V was the rocket developed by the German rocket scientist and aerospace engineering Wernher von Braun. It was developed to support the Apollo program for human exploration of the Moon and was later used to launch Skylab. As of 2020, the Saturn V remains the tallest, heaviest, and most powerful (highest total impulse) rocket ever brought to operational status, and holds records for the heaviest payload launched and largest payload capacity to low Earth orbit (LEO) of 140,000 kg.

(a) Left hand (b) Right hand

Fig. 1: Example frame of the application

I decide to use the Saturn V as main subject of the application because it's a very famous object and and for this there are a lot of free 3D models easily available on the web. This is also true for object linked to the Saturn V like the LEM (Lunar Excursion Module, also known as Apollo Lunar Module) and the Skylab, the first American space station[2].

II. Deterministic Python Script

In this section I will discuss deeply how the deterministic python script work. All the code is freely available on GitHub.

A. Low level script

The deterministic script is a modify and improved version of the script provided by [1].

The original script only check if you close the hand to send a command to Unity (more info on this last part in II-B). I improve the script to obtain a basic level of hand tracking and gesture recognition. In my version I track the hand in the left side of the screen and I use it as a controller (like the mouse for the arrow). The hand in the right side instead is used as a *'left button of the mouse'* i.e. when I close that hand the script send a command to Unity. Two frame of the application are visible in image 1. More precisely in image 1a I shown the tracking of the left hand, with the pink point that represent the *'arrow'*. Image 1b shown a frame with the right hand open.

To better manage hand tracking and gesture recognition I divide the frame in 3 equals part and use the left one and the right one. The central part is ignored and when you used this script your body should occupy this central part.

†University of Padua, {alberto.zancanaro.1}@studenti.unipd.it

[1]It was especially useful the guide and the code provided by [1].

[2]I also select this objects because I have a great passion for space exploration.

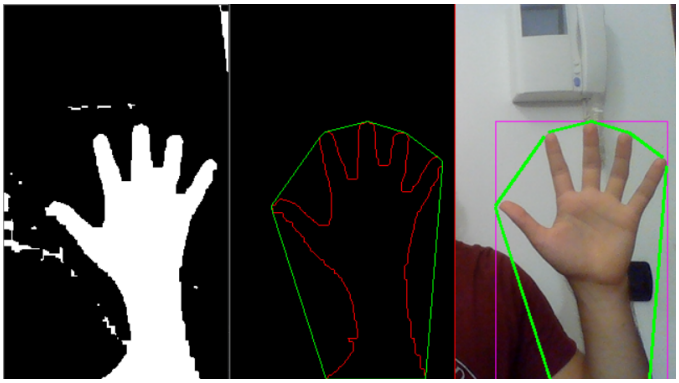


Fig. 2: Stage of the detects hand algorithm

1) Hand Detection: To detect the hand I apply the following step to the frame:

1) Denoise the image with a gauassian filter of kernel 3 x 3.
2) Convert the image to HSV color space.
3) Apply a threshold in the HSV color space to obtain a binary image. All the skin pixel will become white and the others will become black.
4) Apply a dilation and erosion to fill possible void.
5) Reapply a gaussian filter to smooth the edge.

Then I use the OpenCV function *findContours()* to obtain the contours from the binary image. The algorithm used by the function is the one developed by Suzuky and Abe in 1985 ([5]). After that I the Sklansky algorithm ([6]) to find the convex hull from the contours. The Sklansky algorithm was implemented through the OpenCV function *convexHull()*.

In image 2 you can see the steps described above. The left image is the binary image obtain after step 5. The central one is obtained after the application of *findCountours()* and *convexHull()*[3]. The right one is simple the overlay of the central one and of the original frame to show the convex hull in the original image[4].

These steps are the same for both left and right frame.

2) Hand Tracking (Left Side): To use the hand as *'mouse'* I simply decide to put the point in the highest point of the contour found previously. The highest point is defined as the point with the minor y since the the coordinate (0,0) correspond to the upper left corner.

Keeping the index finger raised the highest point will automatically be the index fingertip. This can be seen in image 3.

Due to the noise background the highest point tend to oscillate around the fingertip. To avoid this annoying behavior I implement a stabilizer routine. I always keep the position of the old highest point; when I have a new highest point I evaluate the distance between the new and the old point and I update the position of the point only if the distance is bigger than a certain threshold.

3) Gesture Recognition (Right Side): The gesture recognition is based on the number of fingers of the right hand in each frame. I use a list of 5 elements containing the number of finger detected in the last 5 frame. If the list contain 5 and the actual count of fingers is 0 the script sends a command. The program could also send different command, for example, If I have 3 finger raised and then I close the hand or 5 fingers raised and then I closed the hand

[3]The contours are in red and the convex hull in green.

[4]Convex hull in green.

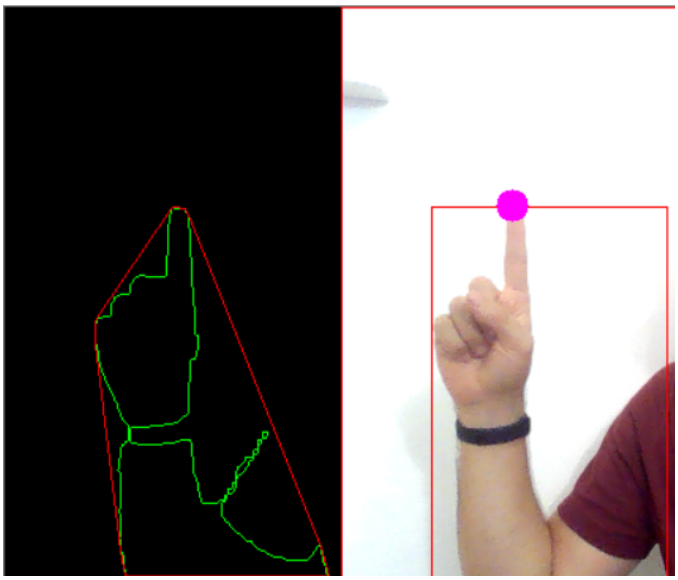
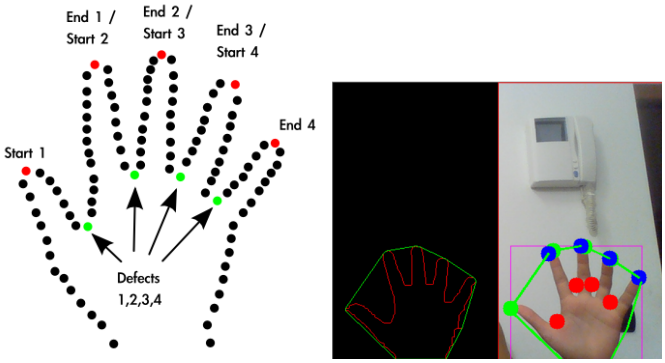


Fig. 3: Highest point



(a) Hand Defects (Scheme) (b) Hand Defects (Script)

Fig. 4: Example frame of the application

The fingers are detected using the cosine theorem. From the convex hull and the contour I find the defects in the hand (figure 4a with the OpenCV function *convexityDefects()*. After that for each defect I form a triangle, triplet of green, blue and red point in image 4b, and evaluate the angle in the red point (that correspond to the defects). To evaluate the angle I use the cosine theorem:

$$\theta = arcos(\frac{b^2 + c^2 - a^2}{2 * b * c}) * \frac{180}{\pi}$$

where θ is the angle in degree and a, b, c the side of one of the triangle defined by the triplet of point in image 4b.

If the angle is less than 90 degree I increase by 1 the counter of fingers for the current frame.

Since I count the defects this method works well only for 3,4 or 5 fingers raised. Also the list doesn't contain the number of actual finger raised but the number of defects counted so for 5 fingers raised I will have 4 defects. The initial explanation of how the script was done with the number of fingers for the sake of simplicity.

B. Unity Interaction

To allow the Python script to communicate with the Unity application I use the UDP protocol.

I pretend that my Python script is on a server and I sent string through UDP to a specific port in my computer (more precisely I use the port number 5065). To do this I use the Python module *socket*. The string that I send are the command for the Unity application that waits for messages from that specific port.

The command that I send at every iteration of the code is the position of the left hand. Also every time I detect the closure of right hand I send an additional command.

C. Pros and Cons

The major advantage of this script is that use of deterministic algorithms allow the script to be light and fast to execute.

At the same time the use of this type of algorithm brings several problems:

- The results of hand tracking are deeply linked to the background and general illumination.
- The results of fingers counter are deeply linked to the background and general illumination.
- Skin segmentation can caused problem.
- The division of the screen allow only basic interactions.

Some example of this problem can be seen in image 5

A possible (not implemented) solution for improve the detection performance is to use Haar filter to detect the face and use the color of the pixel in the face to obtain a more precise threshold to use in HSV color space. This can compensate to slightly change in illumination and skin color.

III. Neural Networks Script

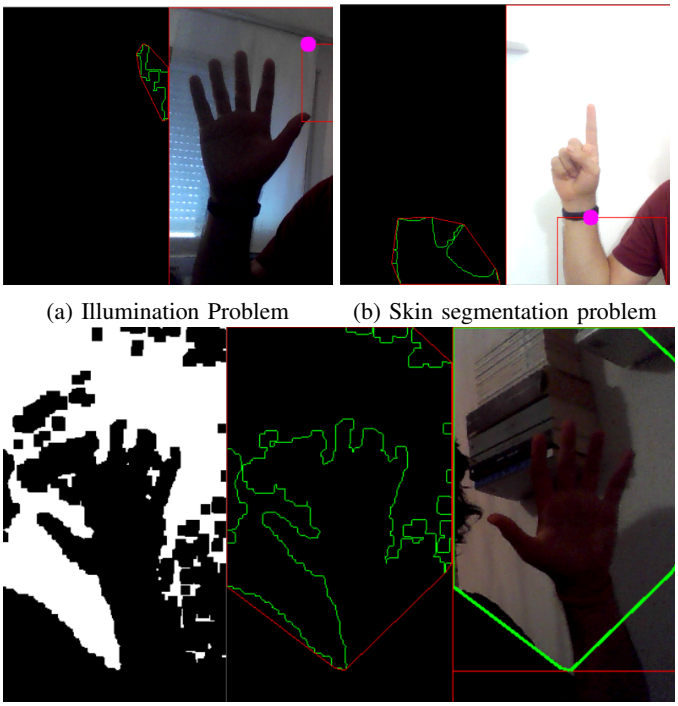
In this section I will discuss how I improve the performance of the hand tracking algorithm and the finger counter algorithm with the use of neural network. Both networks are developed in Pytorch. All the code is freely available on GitHub.

A. Hand Tracking with RCNN

To improve the hand tracking I use a RCNN, a particular type of convolutional neural network used to find various object inside an image. More precisely I used a fast-RCNN[5].

Since this kind of network are very heavy and slow to train I use https://en.wikipedia.org/wiki/Transfer˙learning to retrain a previously trained network. This technique allow to retrain only the last layer of a networks, based on the assumption that the first layer perform always similar operation and extract always similar information from the data. This technique is particularly useful since allow everyone to use network like AlexNet or ResNet that otherwise will be impossibly to train on normal computer.

[5]A particular type of RCNN with improved performance.



(a) Illumination Problem (b) Skin segmentation problem

(c) Noise background problem

Fig. 5: Example of problem with the script

A useful tutorial in Pytorch for transfer learning is provided by [7]. In fact my code was a deeply modification of the original code provided in that tutorial.

The tutorial show how to readapt a mask-RCNN[6]. Since I was only interested in obtain a box around the hand I use a normal RCNN instead of the mask-RCNN. Also I rewrite the entire dataset class to allowed ti be used with my datasets.

I train the network with two different dataset: the EgoHand dataset [8][7] and a custom dataset that I create[8]. Samples of both dataset can be found on image 6.

1) EgoHands Dataset: This dataset contains 48 Google Glass videos of complex, first-person interactions between two people. Between the main feature of the dataset there is:

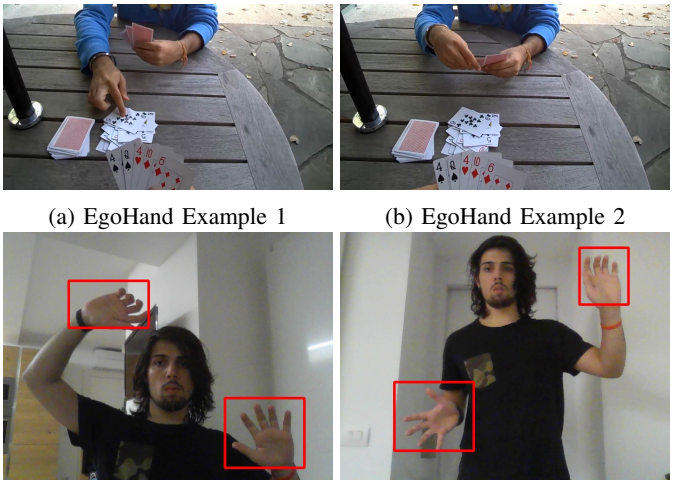
- High quality, pixel-level segmentations of hands.
- The possibility to semantically distinguish between the observer's hands and someone else's hands, as well as left and right hands.

2) MyDataset: This dataset was created through a simple script in Python. Through OpenCV I take the input of the camera and randomly spawn n boxes; after that you have x seconds to position your hand inside the boxes. When the x second finish the script save the frame (without the boxes) in

[6]A particular type of RCNN that beside find the boxes around the object also create a binary mask with their shape.

[7]I use the 1.3 GB versin

[8]Called *MyDataset*.



(a) EgoHand Example 1 (b) EgoHand Example 2

(c) MyDataset Example 1 (d) MyDataset Example 1

Fig. 6: Example samples of the two dataset

a jpg file and the position of the boxes in a npy file. The value of n and x can be decided by the user at the beginning of the script.

If you set $x = 4$ in 10 minutes you already have 150 entry. To train the network I used 420 images.

3) Training and results: The dataset class are extensions of the *torch.utils.data.Dataset*. In this way they can be used together with the Pytorch Dataloader to improve performance during training. To improve memory management the dataset class also store inside them only the path to image and data; in this way when a particularly entry of the dataset is needed is read on fly. This is especially useful to avoid GPU memory problem during training.

The network was trained for 40 epoch with a SGD optimizer. I use a GTX 2070 with CUDA to speed up the train. In the end the results for both dataset were very similar so I decide to use the network trained with my dataset.

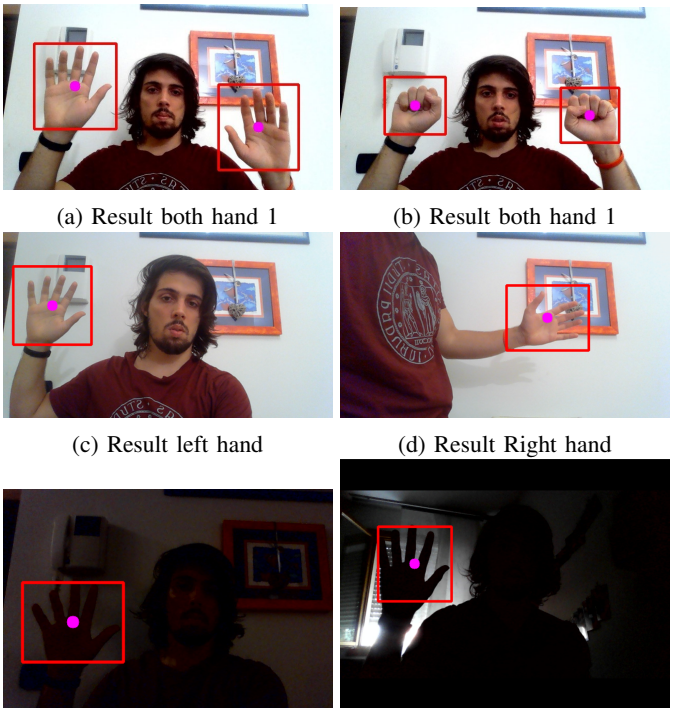
Some results can be seen in image 7. Particularly significant are the result in image 7e and 7f that shown that the network work even in condition of poor lighting or with non uniform background. The image 7a and 7b also shown the ability of the network to track two hand simultaneously.

Since in this case I have the a rectangular box around the hand I decided to use the center of this box as pointer.

B. Fingers counter with CNN

To improve the finger counter I develop a simple CNN that receive in input a hand image and output the number of fingers raised. The architecture is pretty standard with 4 convolutional layer, a max pool layer after each of these and two fully connected layer in the end. ReLU was chosen as the activation function.

The dataset for the training was created with a script that work in similar way to the script described in III-A2. OpenCV capture the image from the camera and display a box of size 140 x 140 at a random position. You have to position your hand inside the box. and after x second the script save the



(a) Result both hand 1 (b) Result both hand 1

(c) Result left hand (d) Result Right hand

(e) Result dark scene 1 (f) Result dark scene 2

Fig. 7: Hand Tracking with the RCNN

hand image[9] and the number of fingers raised in that image. The number of fingers is randomly decided between 0 and 5 and it is shown on screen before take the picture.

I train the network for 250 epoch on 200 images with Cross Entropy as loss function and AdaDelta as optimizer.

The results are more stable respect the method with deterministic algorithm but is still imprecise. This is probably due to a too small dataset. With 500 or 600 images after the train I will probably obtain better results.

C. Unity Interactions

See section II-B. The basic principle is the same but I change the syntax of the command since now I can have two hands, with two different position.

D. Pros e Cons

The major advantage of this implementation is its robustness and its capability to track hands precisely in every environment.

The major disadvantage is the necessity of a GPU to execute the code in real time. To improve performance I used a little trick. Instead of feed every frame into the RCNN I do it every 3 frame. In this way the code works smoothly granted that you had a GPU capable of executing RCNN in relative small time.

In case you have only a CPU the network is too heavy to be executed in a smooth way. If executed on a CPU it caused lag and momentary freezes.

[9]N.B. Saves only the hand image, not the entire frame

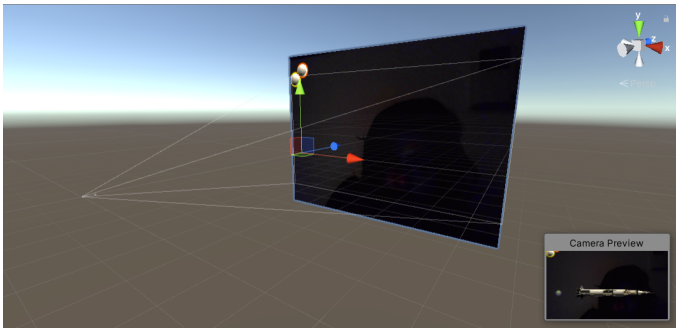


Fig. 8: Vuforia Camera

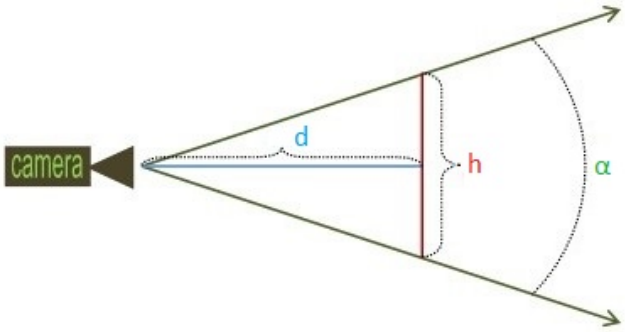


Fig. 9: Field of View

IV. Unity Application

The unity application was a simple augmented reality application to shown how python and Unity can Interact.

All 3D model are free and without copyright.

The application is divided in 2 scene: the first is based on the deterministic script and the second on the CNN script. I upload two short video demo on YouTube; this is the link for the first scene and this is the link for the second one.

Since both Unity and the python script use the camera you need a software to split the camera video stream. I suggest you to use Split Cam.

A. Scene 1

The scene 1 was developed to show how Unity can interact with python.

Like I explain in II-B the python script send command to Unity through the UDP protocol. This command are simple string that contain various information. Ad iteration the python script always send the position of finger of the left hand. Also if detect that you close your hand it send an additional command of *Execute action*.

1) Move pointer: The trickiest part was related the coordinate of the pointer in the python script with the coordinate of unity world. In python I had a Cartesian system with the point (0,0) in the upper left corner. In Unity I have a 3D Cartesian system with (0,0,0) that represent the origin of the world.

From image 8 we can see that the Vuforia camera can be see as a real camera with relative horizontal and vertical field of view. Also we can see that our image is projected in a background plane at distance z from the origin. From figure 9, and from the theory of the field of view, we can derive we value of h as:

$$h = 2d * tan(\frac{\alpha}{2})$$

In our case d is the distance of the background plane from the origin (the coordinate z), h is the x or y coordinate and α is the horizontal or vertical field of view. In this way I can evaluate the maximum x and y coordinate of the background edge. I call this value x_{max} and y_{max}

Once obtained this distance I divide them for the screen height and width. In this way I obtain two value that we call x_{tick} and y_{tick}. With this value I can easily related the the camera coordinate with the unity coordinate with the formula:

$$\begin{cases} x_{unity} = (-x_{max}) + x_{camera} * x_{tick} \\ y_{unity} = (-y_{max}) + y_{camera} * y_{tick} \end{cases}$$

At every iteration of the python script the coordinate of the pointer are send to the Unity application. In the Unity application they are received by a C# script that convert them in the the Unity world coordinate[10]. After that the pointer in Unity is set to that coordinate.

2) Gesture Recognition: When I close the hand the python script send a string with the value *'ACTION'*. If this string is received and the pointer is over one of the button then you are moved to the relative section[11]. To test the application some mouse control were inserted in the button *PAYLOAD*, *BACK* and *LEM*.

Some screen of the application can be seen in image 10 and 11. There's also a YouTube video available at this link.

B. Scene 2

The scene 2 was developed to show a possible application of the use of two hand in an augmented reality application. More precisely I rotate an object with the movement of the hand hand.

The pointer are move in the same way of the scene 1. When I click on the *A* button I detach the Saturn V from horizontal position and it will be aligned with the line that pass for the two pointer. In this way when I rotate my hands the Saturn will follow their rotation.

Some screen of the application are shown in image 12 and 13. There's also a YouTube video at the following link.

V. Results and Conclusion

A. What I learn

From this project I learn how to use Unity, especially the object manipulation through script, the management of Unity transform object and the interaction of Unity with other application. I also better understand how Unity script works and I deepened my knowledge of C#.

[10]Some little misalignment between the python pointer and the Unity pointer can be caused by the different aspect ration of the camera. This can easily fixed adding an offset to both x and y.

[11]E.g. if you are over engine and you close your hand you will be sent to the engine section

Fig. 10: Scene 1 Example 1

Fig. 11: Scene 1 Example 2

Fig. 12: Scene 2 Example 1

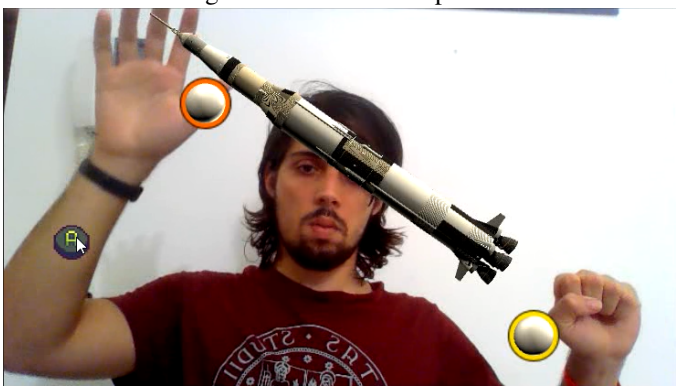


Fig. 13: Scene 2 Example 2

The implementation of neural network was useful for reviewing the use of PyTorch dataset and dataloader. It also allow me to practice with transfer learning, test complicate neural network and to become aware of the TorchVision repository of pre trainer model.

B. Future Improvement

A major improvement will be developed of new application to use the multiple hand tracking with the RCNN. Another upgrade will be the improvement of the finger counter CNN with a better training.

It also worth to study a possible combination between hand tracking with RCNN and finger counter with deterministic algorithm.

Since the EgoHands dataset provide a precise contour of the hand a future variation can be use that contour has a mask and train a mask-RCNN to find precisely the hand instead of find a box around them.

REFERENCES

[1] G. R. Singh, "Introduction to using opencv with unity."
[2] R. Girshick, "Fast r-cnn," 2015.
[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2015.
[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2013.
[5] S. Suzuki and K. Abe, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, pp. 32–46, Apr. 1985.
[6] J. Sklansky, "Finding the convex hull of a simple polygon," *Pattern Recognition Letters*, vol. 1, pp. 79–83, Dec. 1982.
[7] Pytorch, "Torchvision object detection finetuning tutorial."
[8] S. Bambach, S. Lee, D. J. Crandall, and C. Yu, "Lending a hand: Detecting hands and recognizing activities in complex egocentric interactions," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.