# 3D tracking of people from multiple RGBD cameras in an operation room

**Advanced Computer Vision Project**

**Autors :** Monsarrat Samuel & Md Zilan Uddin Saif

**Professor:** Joan Aranda

**Course:** 2022-2023

**Date:** July 29, 2024

ETSEIB
Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

UPC

# 1 Introduction

The development of multi-camera systems has greatly broadened the possibilities of computer vision, allowing for advanced applications such as 3D reconstruction, immersive virtual reality experiences, and accurate object tracking. An essential obstacle in implementing such systems is attaining precise calibration across numerous cameras to guarantee that spatial and temporal data from different sources come together to form a cohesive and coherent picture of the environment. Accurate monitoring of medical professionals is essential in an operating room setting for a range of purposes, including as analyzing workflow, ensuring safety, and providing augmented reality support. The objective of this project is to create a reliable system for tracking individuals in 3D utilizing several RGBD cameras. Two methods are employed : the first uses Deep Neural Networks (DNN) with a pre-trained Single Shot MultiBox Detector (SSD) model for face detection and dlib for facial landmark detection, while the second utilizes T and I pattern detection together with an Histogram of Oriented Gradients (HOG) descriptor for pattern recognition. These methods ensure precise real-time tracking and calibration, creating a robust and adaptable system for operating room environments. Our goal is to provide a dependable and immediate tracking solution by utilizing the depth-sensing features of cameras such as the Intel RealSense, in addition to powerful computer vision and machine learning algorithms.

The methodology employed incorporates various fundamental techniques. Deep Neural Networks (DNN) are used in conjunction with a pre-trained Single Shot MultiBox Detector (SSD) model for face detection, and dlib is used for facial landmark detection. This combination allows real-time detection and precise localization of the head and facial features, essential for accurate movement tracking. The Integral Image approach is utilized to compute the cumulative sum of pixel values inside sub-regions of an image efficiently, reducing the processing burden and enabling speedy recognition of patterns like the "T" and "I" forms created by the head and torso. The Histogram of Oriented Gradients (HOG) descriptor records the distribution of gradient orientations (edge directions) in specific parts of an image, ensuring that identified patterns match a human head by comparing them to a pre-established template. A socket-based communication system manages data from multiple cameras, enabling efficient transfer of tracking data between cameras and the central processing unit, ensuring real-time processing and synchronization. Precise calibration is achieved using techniques outlined by Berthold K. P. Horn, which involves calculating the transformation matrix that relates several camera perspectives using the trajectory of an individual's head. This method employs a matching algorithm to ensure accurate correspondence of each monitored skeleton across multiple camera angles for many individuals.

This comprehensive strategy ensures the tracking system is accurate and adaptable, capable of handling the dynamic and complex environment of an operating room. The project lays the groundwork for improved real-time tracking applications in healthcare by combining deep learning, advanced computer vision algorithms, and reliable data communication protocols. A reliable and effective approach has been created.

# 2 State of the Art Review (SoA)

The tracking of multi-camera systems is a pivotal aspect of modern computer vision applications. Notably, depth-sensing technologies like Intel RealSense have introduced new possibilities for capturing detailed 3D spatial data, enabling more precise camera alignments. Machine learning algorithms have also been increasingly applied to predict and correct calibration and

tracking errors, thereby enhancing the overall system robustness and accuracy. Despite these advancements, the dynamic nature of real-world environments presents ongoing challenges, particularly in terms of real-time processing.

As mentioned earlier, the primary objective of this project is to track the position of individuals. The aim is to cover a broad area using different cameras. Tracking involves monitoring specific points on a person, forming a skeleton. The tracking process is elucidated in the article of Michael Otto et al., "Presenting a Holistic Framework for Scalable, Marker-less Motion Capturing: Skeletal Tracking Performance Analysis, Sensor Fusion Algorithms and Usage in Automotive Industry".

In our specific case, the tracking focuses on the person's head point, the only point essential for this project. The trajectory under analysis will be that of the head. Then, in a second part the trajectory displayed by both cameras have to be displayed in one single coordinate frame. The method chosen to solve this problem is detailed by Berthold K. P. Horn, "Closed-form solution of absolute orientation". As highlighted in the article's introduction, this solution employs a least squares method for three or more points to establish the relationship between two coordinate systems. The methodology is expounded upon in the second part of the article, specifically addressing two orthonormal Cartesian coordinate systems. For more frame systems, the operation is iteratively applied.

Regarding the evolution to multiple person tracking, the problem have been solved by Marco Carraro et al., "Real-time marker-less multi-person 3D pose estimation in RGB-Depth camera networks".

The head detection with hog have been done by Ahamed et al., "HOG-CNN Based Real Time Face Recognition", together with a cnn alogrithm. The paper Sultana et al., "Object Detection using Template and HOG Feature Matching", explain the method uses for head tracking in the case of object recognition.

Kang et al. created a low-power CPU for 3D face frontalization in Deep Learning-based face identification for mobile applications Kang et al., "Low-Power Processor for 3D Face Frontalization in Mobile Devices". The SPHORB algorithm by Vinay A et al. excelled in 2D and 3D facial recognition in robotic surveillance systems. Using ORL and YALE-B datasets, K. Shailaja and colleagues got encouraging results with Adaptive Linear Collaborative Discriminant Regression Classification and Deep Learning. Md Fazlay Rabbi et al. improved smartphone real-time facial recognition with SMARTLET cloudlet work allocation. The HOG-CNN model, using Histogram of Oriented Gradients and Convolutional Neural Networks, enhances accuracy under diverse scenarios Rabbi et al., "SMARTLET: Real-Time Face Recognition on Smartphones with Cloudlet Optimization".

Due to room reflection complexity, linear and spline interpolation and parametric filter modeling were inadequate for BRIR modeling and interpolation. Richard et al. computed and interpolated head rotation HRIRs using a deep neural network, adding head translations and a frequency-domain method to reduce data redundancy and processing expenses. Ren et al. reviewed deep neural networks and a monocular camera for head position recognition, showing the limits of appearance- and model-based methods. A real-time deep neural network architecture was presented for accurate head position prediction with little computational power Ren et al., "Real-Time Head Orientation from a Monocular Camera Using Deep Neural Network".

# 3 Proposed Methodology

## 3.1 System Architecture

The proposed system consists of an array of depth-sensing cameras, including the Intel RealSense d435i, to capture both color (RGB) and depth information. These cameras are positioned to cover different zones, allowing for continuous tracking of one or multiple persons within a single coordinate frame.

## 3.2 Real-time Head Tracking Using Deep Learning Techniques

An algorithm is presented for real-time head tracking using deep learning-based face detection and dlib for facial landmarks detection. The process involves detecting the face, identifying eye centers, and calculating the head coordinates based on the face's position and orientation.

### Deep Neural Network (DNN) for Head Detection

The pre-trained SSD model was used for face detection to obtain the following results. The SSD model is an object detection framework that discretizes the output space of bounding boxes into boxes of different sizes, aspect ratios, and scales for each location on the feature map.

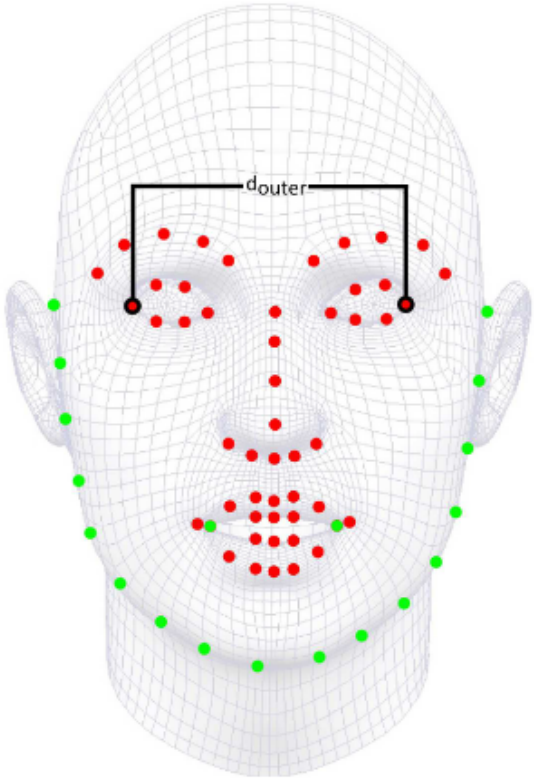



Figure 1: Facial Landmark

## Key Equations and Concepts

- **Input Preprocessing**: The input image is preprocessed by resizing and normalizing it into a blob:

  $$\text{blob} = \text{cv2.dnn.blobFromImage}(\text{cv2.resize}(\text{image}, (100, 100)), 1.0, (100, 100), [104, 117, 123], \text{False}, \text{Fals}$$

- **Forward Pass through the Network**: The preprocessed image blob is passed through the DNN:

  $$\text{detections} = \text{net.forward}()$$

- **Bounding Box Calculation**: The bounding box coordinates are extracted from the detections and scaled back to the size of the original image:

  $$\text{box} = \text{detections}[0, 0, i, 3:7] \times \text{np.array}([\text{frameWidth}, \text{frameHeight}, \text{frameWidth}, \text{frameHeight}])$$

- **Confidence Score**: The confidence score for the detected face is extracted from the detections:

  $$\text{confidence} = \text{detections}[0, 0, i, 2]$$

### Facial Landmarks Detection using dlib

After detecting the face, the code uses dlib's pre-trained 68 facial landmarks predictor to locate specific points on the face, including the eyes.

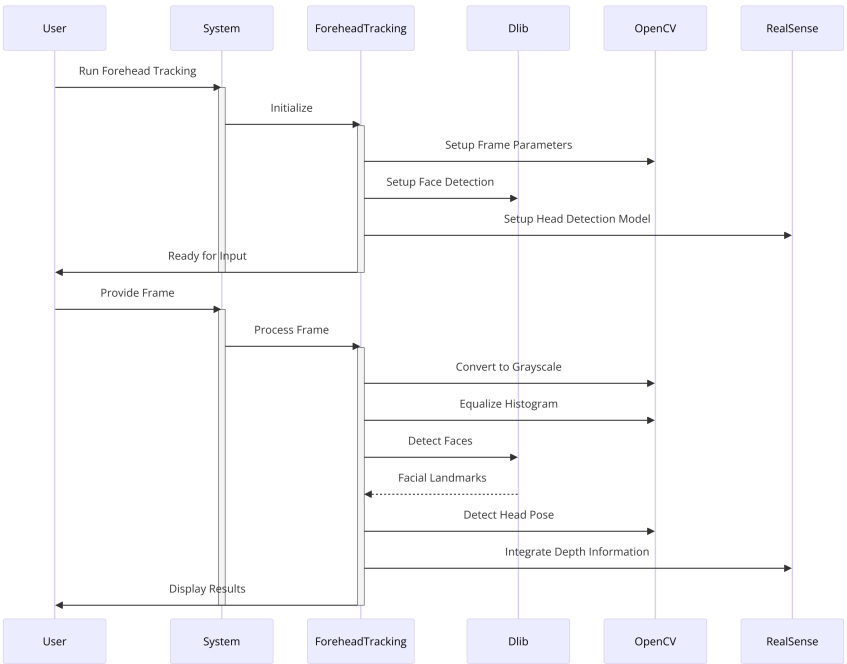



Figure 2: Overall DNN Architecture

### Key Equations and Concepts

- **Eye Center Calculation**: The eye center is calculated by averaging the coordinates of the landmarks corresponding to each eye:

$$(\text{eye\_center}_x, \text{eye\_center}_y) = \left( \frac{\sum \text{shape.part(i).x}}{n}, \frac{\sum \text{shape.part(i).y}}{n} \right) \quad \forall i \in \text{EYE\_INDICES}$$

## Head Region Calculation

The Head region is calculated based on the detected face bounding box and the orientation of the eyes.

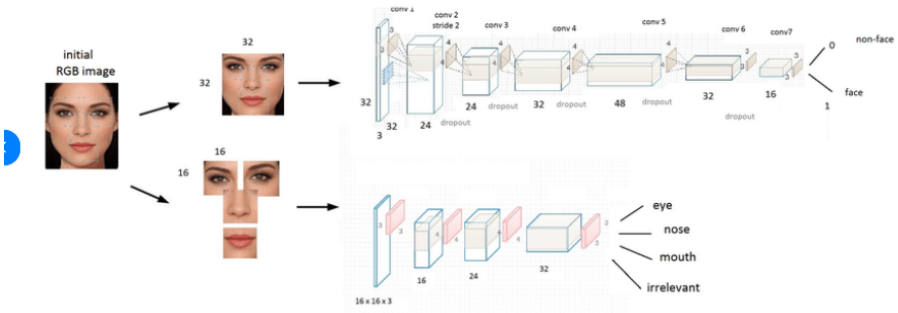



Figure 3: Architecture of DNN

### Key Equations and Concepts

- **Subface Coordinates Calculation**: The coordinates of a subregion of the face (e.g., the Head) are calculated using given ratios:

$$\text{Head\_coord} = \left[ \text{face\_startX} + \text{face\_width} \times \text{ratio}_x - \frac{\text{face\_width} \times \text{ratio}_w}{2}, \ldots \right]$$

- **Eye Angle Calculation**: The angle between the eyes is calculated to adjust the Head region based on the head tilt:

$$\tan(\theta) = \frac{\text{right\_eye\_y} - \text{left\_eye\_y}}{\text{right\_eye\_x} - \text{left\_eye\_x}}$$

$$\theta = \arctan(\tan(\theta))$$

- **Rotation and Translation**: The points are rotated and translated to align the Head region correctly:

$$\text{rotated\_point} = (x\cos(\theta) - y\sin(\theta), x\sin(\theta) + y\cos(\theta))$$

$$\text{translated\_point} = (\text{rotated\_point}_x + \text{offset}_x, \text{rotated\_point}_y + \text{offset}_y)$$

## Gradient Calculation and Image Transformation

The Head region is extracted and aligned correctly using perspective transformation.

**Key Equations and Concepts**

- **Perspective Transformation Matrix**: The perspective transformation matrix is calculated to warp the image region:

$$M = \text{cv2.getPerspectiveTransform}(\text{src\_pts}, \text{dst\_pts})$$

- **Warp Perspective**: The image is warped to align the Head region based on the calculated transformation matrix:

$$\text{warped\_image} = \text{cv2.warpPerspective}(\text{self.frame\_in}, M, (\text{self.frameWidth}, \text{self.frameHeight}))$$

Deep Neural Networks (DNN) are utilized for face detection and dlib for detecting facial landmarks to monitor the head and forehead in real-time. A face is detected using an SSD model, and precise facial landmarks are determined to establish the positions of the eyes and the forehead area. The frontal area is subsequently adjusted according to the orientation of the head through rotation and translation modifications. This method ensures precise monitoring of the forehead, even with head movements. The T-pattern detection method employs integral images to rapidly calculate the sums of pixels within rectangular areas, enabling fast recognition of specific patterns based on their structural attributes. The system efficiently recognizes and tracks facial characteristics and patterns in real-time by utilizing a combination of gradient-based HOG features and integral image approaches. This implementation demonstrates the effectiveness of integrating deep learning and computer vision techniques for real-time head tracking. The application of DNNs for face detection combined with the precise localization of facial landmarks provides a robust foundation for further development.

### 3.3 Head tracking using T and I pattern with Intel Realsense depth camera

Advanced computer vision techniques are leveraged to detect the "T" pattern, compute Histogram of Oriented Gradients (HOG) features, and utilize a DNN algorithm to track the head and eyes while handling depth and color streams from an Intel RealSense camera. Each part of the code and the theoretical approach is explained below, including key equations and formulas. The head and torso of a person can be seen as a reverse T pattern when viewed from the front or back and as an I pattern when viewed from the side.

**Integral Image Concept:** An integral image is a data structure that represents high-speed calculations of the sum of pixel values over a rectangular area of an image. The integral image will be used several times in the algorithm, so the computation of an average pixel value for a region will be reduced to a minimum.

**Integral Image Calculation:**

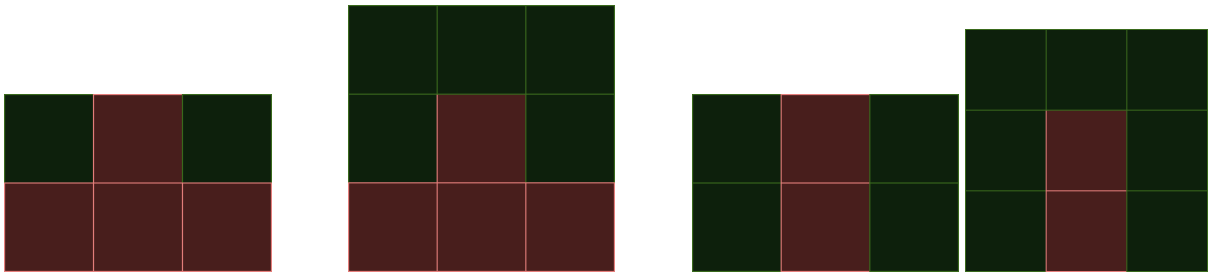
$$\text{integral\_image}(x, y) = \sum_{i=0}^{x} \sum_{j=0}^{y} \text{image}(i, j)$$

**Block Sum Calculation:**

$$\text{Sum of block} = \text{int\_im}(x_1, y_1) + \text{int\_im}(x_2, y_2) - \text{int\_im}(x_1, y_2) - \text{int\_im}(x_2, y_1)$$

**Block Average Calculation:**

$$\text{average\_value} = \frac{\text{Sum of block}}{\text{width} \times \text{height}}$$



(a) T pattern detection using 6 block

(b) T pattern detection using 9 block

(c) I pattern detection using 6 block

(d) I pattern detection using 9 block

Figure 4: Example of pattern tracked by the Algorithm

In the Figure 4 the patterns tracked by the algorithm are displayed, where a red block represent a block where in average the values of the depth image are smaller and a green block where those values are larger. To compare the value of each bloc and decide if there is a pattern or not the following ideas have been apply.

1. Implementation of a Threshold and a number of error All the green block are compared with the red block for each green bloc that have a smaller average value than any red block an error counter is incremented. To consider if the block is greater or not depend on the following equation and is performed by the function 'check_error':

$$Green\_block < Red\_Block * Threshold \tag{1}$$

   As said before, the Green block is supposed to have a greater value than the red block, when this equation is not verified (because Green block have a higher value) than nothing happen, however when the equation is verified than an error is encountered.

2. Implementation of a Score To understand how much the pattern detected correspond to the one the algorithm is searching a score have been implemented. The equation for the score is the following :

$$Score = \sum_{n=1}^{Nbredblock} - \sum_{n=1}^{NbGreenblock} \tag{2}$$

   Where the goal is to minimize the score, this is calculated with the function 'Compute_score' and allow to know the general distance between the green and red blocks.

Hence to decide if a block correspond to a pattern or not, 3 parameter can be tuned : the number of error allowed, the threshold between block and use or not the best scores. To be sure that the detected pattern correspond to a head they can be compared to a template using histogram of gradients as explained in the following section.

## 3.4 Histogram of Oriented Gradients (HOG) Descriptor

With potential heads identified by the T and I pattern detector, a recognition tool is required to confirm that the tracked area is indeed a head. The Histogram of Oriented Gradients (HOG) descriptor, widely used in computer vision for object detection, serves this purpose. HOG describes the shape and appearance of objects by capturing the distribution of gradient orientations (edge directions) in localized regions of an image.

The HOG descriptor is implemented for pattern recognition using a template. The descriptor is made block by block, at each pixel, the gradient magnitude and direction are calculated using Sobel filters. These gradients are then binned into orientation histograms within small cells, capturing edge information. This process effectively highlights the structural characteristics of objects, making HOG robust for detecting specific patterns like faces and tracking heads. The HOG features are utilized to match and recognize faces and heads by comparing the HOG descriptor of a template image against regions in the target image.

The final HOG descriptor is constructed by concatenating the normalized histograms from all blocks in the image into a single feature vector.

### Gradient Computation

The first step in the HOG descriptor process is to compute the gradient of the image in both the horizontal and vertical directions using Sobel operators. This approach highlights changes in intensity which are crucial for edge detection:

- Sobel filter for the x-direction ($G_x$):

$$\text{filter}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

- Sobel filter for the y-direction ($G_y$):

$$\text{filter}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

These filters are applied to the image to obtain the gradients in the x and y directions, respectively.

#### Gradient Magnitude and Direction

Once had the gradients $G_x$ and $G_y$, calculated the gradient magnitude and direction for each pixel:

$$G = \sqrt{G_x^2 + G_y^2}, \quad \theta = \arctan 2(G_y, G_x)$$

The magnitude represents the strength of the edge, and the direction indicates the orientation of the edge. These gradient measures are essential for constructing the orientation histograms that define the HOG descriptor.

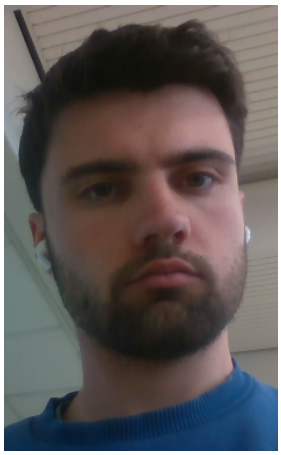
#### Orientation Binning

The image is divided into small spatial regions called cells (e.g., 8x8 pixels). For each cell, we create a histogram of gradient directions. The range of directions (0° to 180°) is divided into bins (e.g., 9 bins, each covering 20°). Each pixel in the cell votes for a bin based on its gradient direction, weighted by the gradient magnitude.

#### Template matching

The HOG is first calculate on a template that represent a human face. Than the algorithm compare the result with another image by using the following formula :

$$Score = \|Hog\_Template - Hog\_img\| \tag{3}$$



(a) Example of Template

(b) Other Example of Template

### 3.5 Socket Implementation

In the case of using multiple cameras, the raw data of those cameras can't be obtain and processes in a single computer. For this reason the communication between computer is essential to gather the data. To do so a socket-based solution have been implemented, it involves setting up a server to receive data from at least 2 clients (presumably 2 cameras) and forward it to another server for further processing.

Two different type of socket have been created to receive the data, one using Wi-fi connection and one with local connection. To send the data to the Matlab script another socket named 'forward socket' have been created. The receivers socket are using the port 8080, and the ip adress correspond either to `10.42.0.1` in case of a Wifi connection either to `127.0.0.1` for a local host. The receiver in this case use ip `0.0.0.0` which mean listening to every ip adress that try to connect on the port 8080.

Additionally, a forwarding socket was established to relay the received data to another server. This socket, `forward_socket` is the socket that connect with matlab. It was created locally hence have ip `127.0.0.1` and use the port 65435.

The server continuously receives data chunks from the client using `data = conn.recv(1024)`. This data is then decoded from bytes to a string and split into individual lines. To format the

received data, the first line is separated from the rest because it represent the **number of the camera**. This value is very important because it's needed to know from which camera the data are coming from. While subsequent lines, which contain face coordinates are processed. Each coordinates is split by commas while each trajectory is splited by a '\n'. This means that for example a message with 2 person detected will look like this : "x1,y1,z1 \n x2,y2,z2".

Once the data is formatted, it is forwarded to the designated server using `forward_socket.sendall`. This step ensures that the processed data reaches its final destination for further handling or analysis. Finally, after the data is forwarded, the client connection is closed with `conn.close()`, and upon termination of the program, both the server and forwarding sockets are closed using `server_socket.close()` and `forward_socket.close()`. This comprehensive approach ensures efficient and reliable transmission of calibration data, leveraging socket programming for real-time applications.

## 3.6 Live calibration of the cameras

### Single person calibration

The head position of the individuals are obtained by each cameras in a relative position. Since both camera are seing the exact same trajectory but with a different point of view, it's possible to obtain the transformation matrix between both cameras if the trajectory contain enough direction changes. This problem is called the absolute orientation problem and have been solved by Berthold K. P. Horn, "Closed-form solution of absolute orientation". The algorithm to calibrate two trajectories is performing the following steps :

1. Centroid computation of both set of points
2. Computation of the relative points
3. Computation of the matrix $\mathbf{M} = \sum_{i=1}^{n} (\mathbf{r}_{1,i} - \mathbf{c}_1)(\mathbf{r}_{2,i} - \mathbf{c}_2)$
4. Computation of the SVD of matrix M to obtain the rotation Matrix $\mathbf{R} = \mathbf{V}\mathbf{U}^T$
5. computation of the translation vector $\mathbf{t} = \mathbf{c}_2 - \mathbf{R} \times \mathbf{c}_1$

As it can be seen, the algorithm needs to compute the SVD of a matrix which have the same size than the number of points. This operation is computationally expensive hence it has been chosen not to perform the calibration every time that a point is received. A new calibration matrix is computed only if the new amount of information is considered sufficient.
The conditions to perform a new calibration are the following :

- The calibration is possible only every 30 points
- The total average relative angle per trajectory in degree since the last calibration is superior to 500.

The last point is used to be sure that the trajectory is not a straight line which will not bring any information to the calibration.

**Mutli person calibration**

The problem can become more complex when multiple individuals are walking towards the cameras. Despite the increased complexity, the fundamental problem remains the same. Now, there is a possibility of having 1, 2, 3, or more persons in the scene. Since only 2 cameras are employed, each skeleton must be paired with exactly one other skeleton. The algorithm randomly selects an unmatched skeleton and applies the same matching algorithm as in the previous section for every unpaired skeleton. The error is computed for each skeleton, and the algorithm selects the skeleton that yields the minimum error. This process is repeated until all the skeletons are matched.

In this case the new calibration is performed every 30 points, not taking into consideration the relative angle of the trajectories.

**Architecture of the Calibration program**

The Calibration Algorithm have been coded in Matlab. The architecture is composed of a File named "Calibration_Decider.m", the role of this algorithm is to decode the data sended by the python script and check the coherence of the data to avoid error during calibration. Afterwards it sended the points to calibrate when the previous conditions are fulfilled. While sending the points, it also send the number of trajectories and the length in bytes of the message. The receiver of this data is the script 'Receiver_Calibrate', adapt the algorithm in function of the number of trajectories, in the case of 2 trajectories it means single person calibration, if there is more it means multi-person calibration. For all the point sended the transformation matrix and the associated error is keeped.

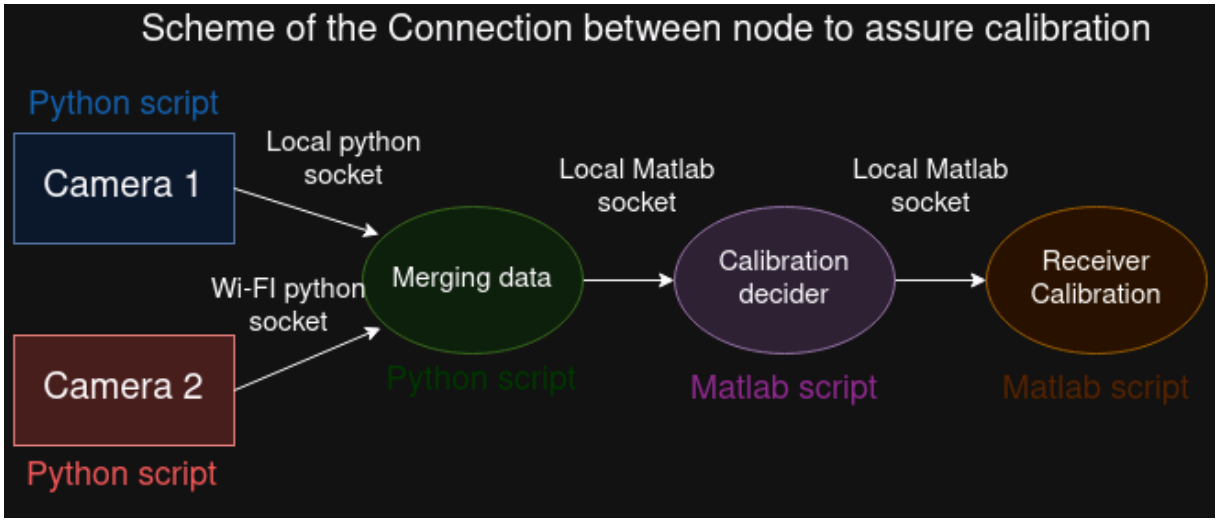



Figure 6: Scheme of the communication architecture to assure calibration

# 4 Results

### Hog matching

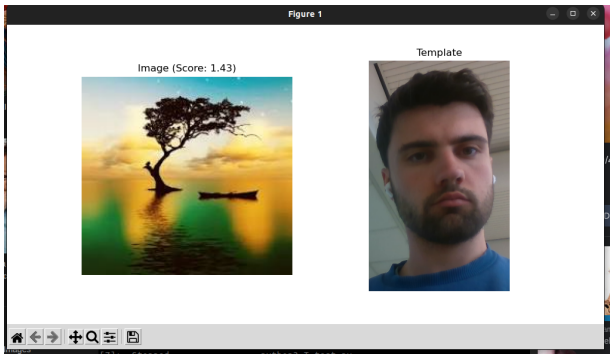



Figure 7: Score of Hog with a random image

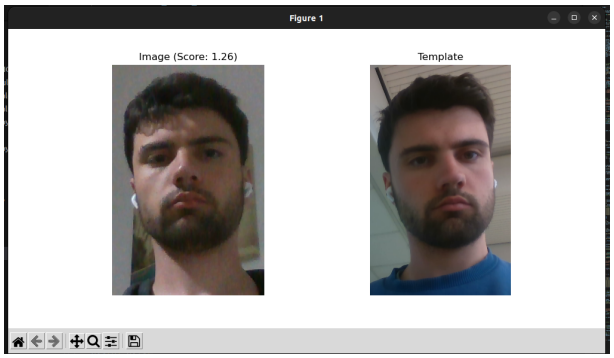
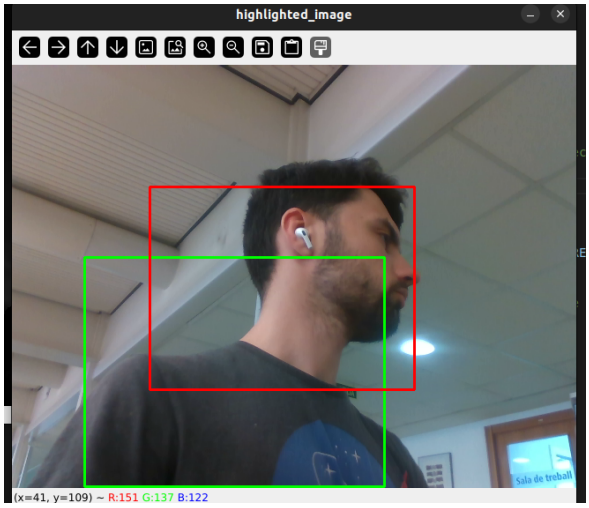



Figure 8: Score of Hog with a face image

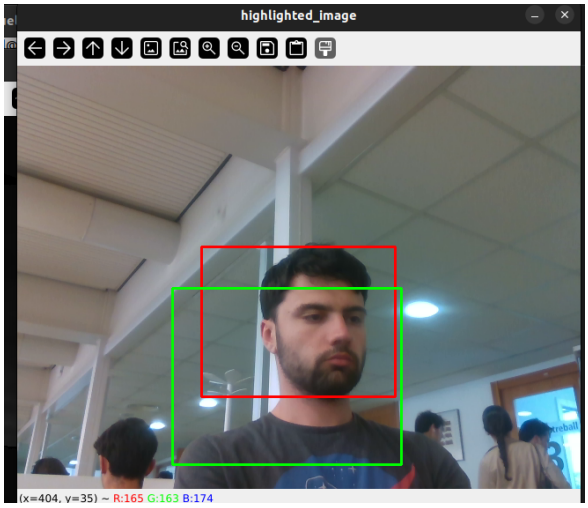
As it can be seen the hog gives a lower score to the image with a face and a higher score to the random which is wanted. In general the hog matching do not always recognize a face but very often assign a high score to a image that do not contain a face, which is wanted. After making experiment, a score containing an image could be around 1.28 or below.

### T and I pattern matching

In the Figure 9 the best I pattern detected is shown in red while the best T pattern detected is shown in green. As it can be seen the T pattern is capturing also the torso which is not fully desirable however in general it has been shown through experiment that the T pattern is less sensible to miss matching.In general to avoid miss matching the hog is used to remove the maximum of those. However it is to notice that the computational time to perform T and I pattern is quite high leading to a certain lag in the video. Moreover using hog is even worse in term of computational time, hence it has been chosens to apply the hog only on the best matches (maximum 9 match).




(a) Result of T pattern 6 block using Hog confirmation Profile view




(b) Result of T pattern 6 block using Hog confirmation Frontal view

Figure 9: General result for T tracking with 6 block

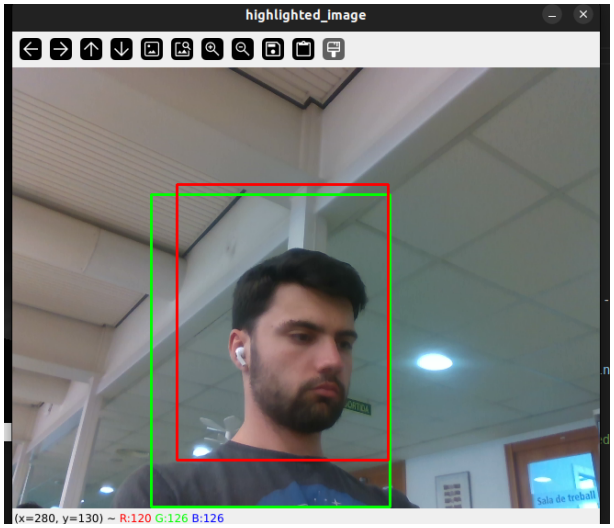
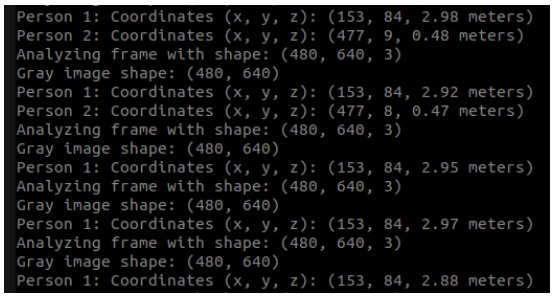



Figure 10: Result of T pattern and I pattern for 9 blocks

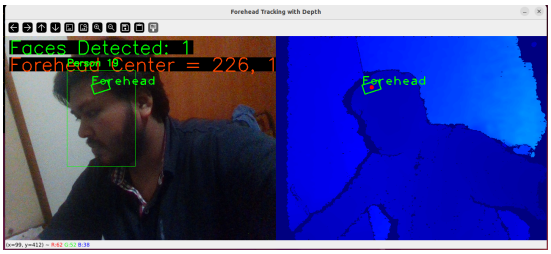
Using a T pattern with 9 block give similar result to the one of 6 blocks however as it can be seen in the Figure 10 the tracking now also include the top of the head.

### DNN algorithm result

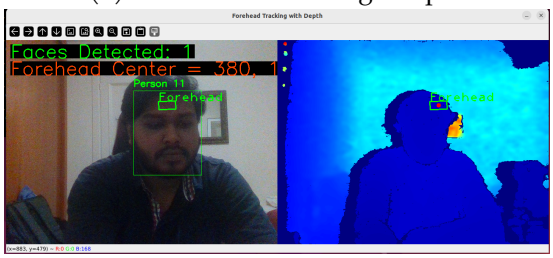
The outcomes obtained by the DNN algorithm showcase its proficiency in real-time monitoring of individuals' head positions utilizing RGBD cameras. In Figure 11, all the results have been shown, including single and multi-tracking of the head in both color and depth images. The algorithm consistently monitors and revises the 3D coordinates (x, y, z) of identified heads, precisely indicating their locations and distances. The system effectively detects faces, indicating their location with green bounding boxes, and accurately identifies the forehead area for exact head tracking. The DNN algorithm utilizes depth information to enable precise 3D placement, keeping consistent performance regardless of individuals' proximity to the camera. In summary, the DNN algorithm demonstrates resilience and efficiency, delivering reliable and precise real-time 3D head tracking. This makes it well-suited for use in dynamic settings such as operation rooms.



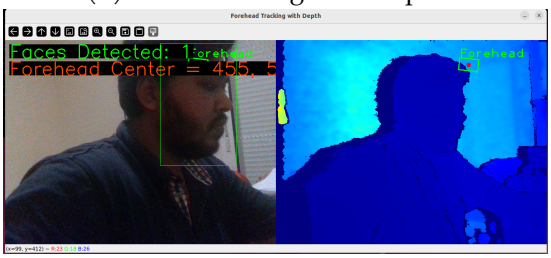

(a) Coordinate Tracking Output




(b) Head Tracking with Depth 1

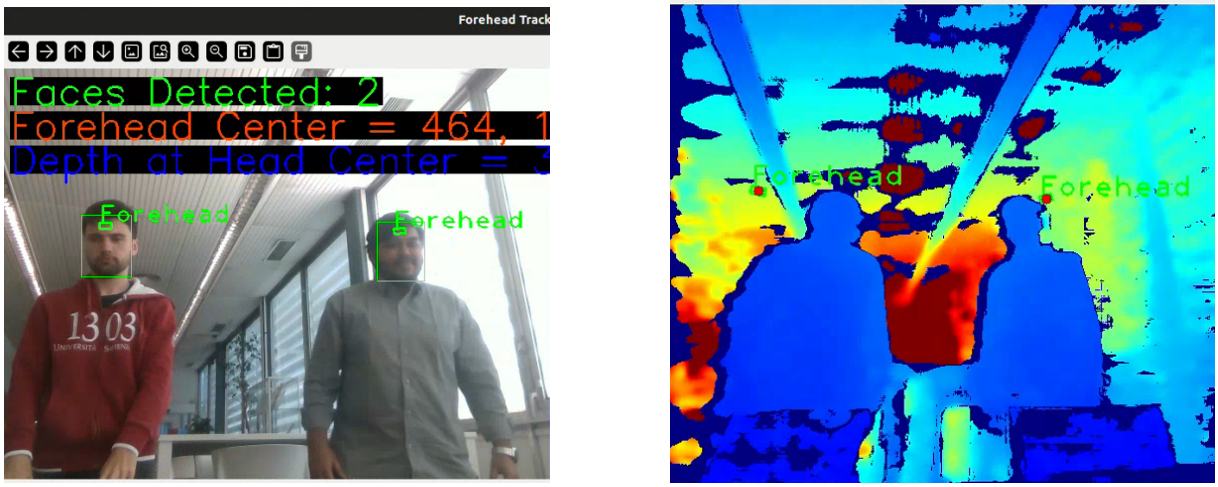



(c) Head Tracking with Depth 2




(d) Head Tracking with Depth 3

Figure 11: Results from the DNN algorithm for real-time head tracking using RGBD cameras for single individual




(a) Multi Head tracking

(b) Multi Head Tracking in depth

Figure 12: Results from the DNN algorithm for real-time head tracking using RGBD cameras for multiple individual

## 5 Prospects for Future Work and Enhancements

The current implementation of the 3D tracking system, which utilizes RGBD cameras and DNN algorithms, demonstrates promising performance in various applications. However, certain aspects require further refinement to enhance the system's accuracy and reliability, particularly in dynamic and complex environments.

A significant concern is the challenge of simultaneously monitoring multiple individuals. The existing system occasionally struggles to maintain consistent labels for individuals, especially when they come into close proximity. This can lead to ambiguity in distinguishing between individuals, resulting in tracking errors. Enhancements could focus on developing more intricate algorithms for multi-person monitoring that maintain consistent labels and accurately differentiate individuals, even in close proximity.

Another area for improvement is the system's management of coordinate adjustments when two individuals cross paths. The x, y, z coordinates may vary, leading to inaccuracies in tracking their exact positions. To address these issues, the stability and resilience of the coordinate tracking algorithm can be enhanced by incorporating advanced filtering techniques such as Kalman filters or utilizing machine learning models that account for spatial dynamics.

Additionally, while the T pattern detection is effective at close range, its performance deteriorates with increasing distance. This limitation affects the system's ability to accurately monitor individuals at greater distances. Further research could explore the development of more sophisticated pattern recognition methods or the integration of supplementary sensor data to improve accuracy at various distances.

By addressing these challenges, the system can achieve higher precision and reliability in live 3D tracking, making it more suitable for dynamic environments such as operating rooms. These improvements will enhance the overall functionality and robustness of the tracking system, increasing its acceptance and applicability.

The outcomes obtained by the DNN algorithm showcase its proficiency in real-time monitoring of individuals' head positions using RGBD cameras. The algorithm consistently tracks and updates the 3D coordinates (x, y, z) of identified heads, accurately indicating their locations and distances. The system effectively detects faces and marks them with green bounding boxes, identifying the forehead region for precise head tracking. The DNN algorithm utilizes depth information to provide accurate 3D placement, ensuring consistent performance regardless of individuals' proximity to the camera. Overall, the DNN algorithm demonstrates resilience and efficiency, delivering reliable and precise real-time 3D head tracking, making it well-suited for dynamic contexts such as operating rooms.

# 6 Conclusion

The project has successfully developed a comprehensive system for tracking humans in a 3D space within an operating room, utilizing multiple RGBD cameras. Through deep learning and advanced computer vision techniques, a reliable and effective methodology for real-time head tracking was established.

The system relies on several key components. Deep Neural Networks (DNN) combined with a pre-trained Single Shot MultiBox Detector (SSD) model for face detection, along with dlib for facial landmarks detection, provided accurate and nearly instantaneous identification of head positions. Integral Image methods enabled rapid computation of pixel value sums within subregions of images, facilitating the identification of specific patterns such as "T" and "I" forms that signify head and torso positions.

The Histogram of Oriented Gradients (HOG) descriptor played a crucial role in detecting and verifying head patterns by capturing the distribution of gradient orientations. This feature ensured that identified regions matched pre-established templates, enhancing the precision and reliability of the tracking system.

Socket programming enabled efficient data transmission between multiple cameras and the central processor unit, ensuring synchronization and real-time processing. Berthold K. P. Horn's calibration method allowed for accurate alignment of camera views, ensuring precise mapping of trajectories from various perspectives. Both single-person and multi-person calibration methods were effectively implemented to accommodate different scenarios within the operating room.

The results demonstrated that the HOG matching algorithm successfully differentiated faces from random images, while the T and I pattern matching consistently recognized head and torso regions. However, optimizing computational time remains essential to achieve higher efficiency.

In summary, this study has shown the potential of combining deep learning and computer vision for real-time tracking applications in healthcare settings. The developed system can significantly improve workflow analysis, safety monitoring, and augmented reality assistance in medical environments. Enhancements in computing efficiency and real-time processing capabilities will further enhance the system's application and performance in the future.

## References

Ahamed, H., Alam, I., & Islam, M. M. (2018). Hog-cnn based real time face recognition. *2018 International Conference on Advancement in Electrical and Electronic Engineering (ICAEEE)*, 1–4. https://doi.org/10.1109/ICAEEE.2018.8642989

Carraro, M., Munaro, M., Burke, J., & Menegatti, E. (2017). Real-time marker-less multi-person 3d pose estimation in rgb-depth camera networks [Available at https://arxiv.org/abs/1710.06235].

Horn, B. K. P. (1986). Closed-form solution of absolute orientation [Available at https://web.stanford.edu/class/cs273/refs/Absolute-OPT.pdf].

Kang, S., Kim, H., & Lee, J. (2018). Low-power processor for 3d face frontalization in mobile devices. *IEEE Transactions on Circuits and Systems for Video Technology*, *28*(5), 1223–1232.

Otto, M. M., Agethen, P., Geiselhart, F., Rietzler, M., Gaisbauer, F., & Rukzio, E. (2017). Presenting a holistic framework for scalable, marker-less motion capturing: Skeletal tracking performance analysis, sensor fusion algorithms and usage in automotive industry [Available at https://www.jvrb.org/past-issues/13.2016/4481].

Rabbi, M. F., Islam, S., & Ahmed, M. (2019). Smartlet: Real-time face recognition on smartphones with cloudlet optimization. *IEEE Transactions on Mobile Computing*, *18*(3), 616–629.

Ren, J., Zhang, Y., & Wang, J. (2019). Real-time head orientation from a monocular camera using deep neural network. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3677–3686.

Sultana, M., Ahmed, T., Chakraborty, P., Khatun, M., Hasan, M. R., & Uddin, M. S. (2020). Object detection using template and hog feature matching. *International Journal of Advanced Computer Science and Applications*, *11*, 233–238. https://doi.org/10.14569/IJACSA.2020.0110730