LAB 4 - HISTOGRAM-BASED OBJECT TRACKING

APPLIED VIDEO SEQUENCES ANALYSIS

Zsófia Sára Budai and Juan Manuel Peña Trapero
{budai9902, juanmaptcg}@gmail.com

I. INTRODUCTION

This laboratory report describes the implementation and analysis of a C++ program for single object tracking based on histograms using the OpenCV library. The objective of the lab is to develop a visual tracker based on color and gradient features, and analyze the strengths and weaknesses of the different modules/features that compose the pipeline of the tracker. In addition, the lab aims to increase the independence and self-learning skills of the students by applying the hybrid pair-programming strategy for combining individual and collaborative work to achieve the goals. The report presents the results of tasks 4.2, 4.3, and 4.4, which involve the implementation and analysis of the color-based and gradient-based trackers on real data. The report also includes an appendix with information on difficulties encountered during individual work, modifications made during collaborative work, time logs for collaborative work, and an overall impression of the hybrid pair programming methodology.

The report is organized as follows: Section 2 presents a brief overview of the underlying theory of visual tracking and histogram matching. Section 3 describes the implementation of the color-based and gradient-based trackers based on the requirements of tasks 4.2, 4.3, and 4.4. This section includes details on the source code, implementation choices, and modifications made from the original paper [1]. Section 4 presents the experimental methodology, analysis, and discussion of results for tasks 4.3 and Section 5 for task 4.4, including visual examples of bad tracking cases. This section also includes tables comparing different parameter settings and performance evaluation metrics. As closure, the Appendix I with information on difficulties encountered during individual work, modifications made during collaborative work, time logs for collaborative work, and an overall impression of the hybrid pair programming methodology. Finally, Section 6 provides a conclusion and suggestions for future work.

II. HISTOGRAM-BASED TRACKING: OVERVIEW

Histogram-based object tracking is a widely used technique in computer vision for tracking objects in video sequences. Based on [1], this approach combines intensity gradients and color histograms to accurately track an object's position and scale. For this lab, the intensity gradients or the fusion is implemented using HOG features but the fusion of both methods is not implemented. The process begins with object initialization in the first frame, followed by feature extraction, where relevant features are extracted from the object. An object model, represented by a set of parameters, is then created and continuously updated using an adaptive scheme to account for appearance changes. The tracking phase involves predicting the object's position in subsequent frames by comparing its appearance with the model. An iterative refinement step enhances tracking accuracy by updating the model and refining the object's position estimation. Overall, histogram-based object tracking offers a robust method for localizing and tracking objects of interest in video sequences.

III. CODE IMPLEMENTATION: MERGING PARTS (TASK4.2)

This section describes the code developed from the combination of the individual work in the first part of this lab. In order to describe the software development, the key functions for tracking and its evaluation are described below:

- **estimateTrackingPerformance** compares two lists of bounding boxes to estimate their overlap using the Intersection over Union (IOU) criterion. The IOU ranges from 0 (worst) to 1 (best) and is computed for each frame. The function takes two parameters: Bbox_GT (a list of ground truth bounding boxes) and Bbox_est (a list of estimated bounding boxes). It returns a list of float values representing the IOU scores for each frame. The function iterates through each frame, calculates the intersection and union of the corresponding bounding boxes, and computes the IOU as the ratio of the intersection area to the union area. The IOU score is then added to the result list.
- **createCandidates** generates a list of candidate bounding boxes around a predicted bounding box. It takes several parameters:
  - numCand (the number of candidates to create)
    - numCand = 1 for 9 candidates
    - numCand = 2 for 25 candidates
    - numCand = 3 for 49 candidates
    - numCand = 4 for 81 candidates
  - pred (the predicted bounding box)
  - w and h (the width and height of the candidate bounding boxes)
  - step (the step size for the grid)
  - frameSize (the size of the frame/image)

  . The function initializes an empty list called candidates. It calculates the step sizes (stepX and stepY) based on the predicted bounding box dimensions. Then, it iterates over a range of values for i and j, creating candidate bounding boxes by offsetting the predicted bounding box's position. Each candidate bounding box is checked to ensure it does not exceed the frame boundaries. Valid candidates are added to the candidates list, and finally, the list is returned.

- **compareCandidates** compares a given histogram with a list of candidate bounding boxes using a specified feature. It takes four parameters: histogram (the histogram to compare), candidates (a list of candidate bounding boxes), feature (a string specifying the feature type), and frame (the input image/frame). The function initializes an empty list called distances to store the comparison results. It then converts the histogram to CV_32F data type. For each candidate bounding box, it computes a histogram (c_histogram) based on the specified feature and color feature. The candidate histogram is also converted to CV_32F data type. The function compares the input histogram with each candidate histogram using the Bhattacharyya distance measure (compareHist with CV_COMP_BHATTACHARYYA). The resulting distance is added to the distances list. Finally, the list of distances is returned.
- **computeHistrogram** computes the histogram of a given input image based on the specified feature and color feature. It takes several parameters: box_image (the input image for which the histogram needs to be computed), out_box_image (a reference to the output image representing the selected feature of the input image), feature (a string specifying the feature type), and color_feature (a character representing the color feature to be considered). The function starts by declaring several variables and matrices. Based on the feature type, the function performs different operations. If the feature is "RGB," the input image is split into color channels, and the appropriate channel is selected based on the color feature. If the feature is "HSV," the input image is converted to HSV color space, split into channels, and the color channel is selected based on the color feature. For "HOG" and "GRAY" features, the input image is converted to grayscale. After selecting the appropriate channel or converting

This group of functions are then used in the main file. The main continuously reads frames from the video source and performs object tracking using a grid-based candidate generation approach and template matching based on histogram comparison. The system generates candidate locations, computes histograms of the region of interest, compares them with candidate histograms, selects the best candidate, and updates the estimated bounding box. The processing time and tracking performance are measured and displayed. Finally, the program releases resources and provides statistics on processing time and tracking performance.

As a summary, the key parameters that are going to be tuned for this lab report are:

- nbins: number of bins of the histogram.
- ncandidates: number of candidates

Once all the code is merged the video sequence "bolt" is used for testing the performance of the final system that is going to be the base for the rest of this laboratory. The performance of the first 150 frames of the sequenced for 16 bins and 81 candidates are throwing decent visual results (the computed bounding box is always almost centered on the ground truth bounding box). One of the remarkable observations on this stage is how relevant the initialization of the bounding box dimension is for the performance of the sequence. If the first frame is selected as a reference for initializing the candidate generation and computing the original histogram of the tracked object, the performance falls bellow 50%. This is due to the size differences of the runner and it ground truth bounding box as it gets closer to the camera and larger on the scene. This situation is shown in the Figure 1. Since our code is not prepared for updating drastically the size or aspect ratio of the bounding box it fails to keep the scale of the predicted bounding box and ends up getting this low performance. In contrast, if the 100th frame is used as a reference the performance improves by almost 15%. Nevertheless it is important to remark that, on a real-world tracking scenario, the ground truth is not going to be available so only the initial bounding box is going to be provided tot he system.

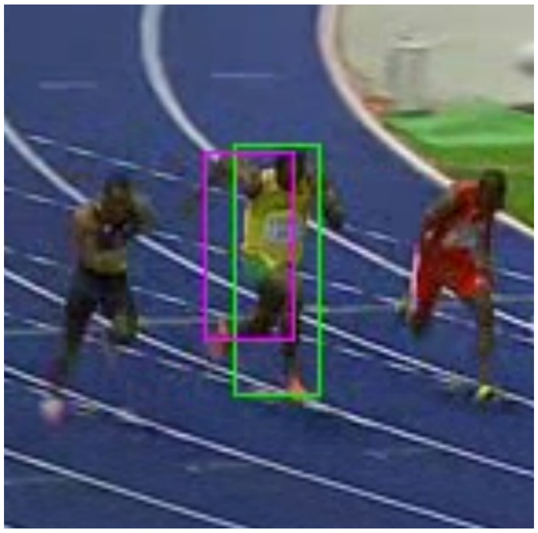

Fig. 1. Crop of frame 46 of the bolt test sequence with ground truth (green) and computed with red histogram (purple) bounding boxes.

IV. Color-based tracking: analysis over real data (Task 4.3)

In this section, the performance of color-based tracking algorithms is evaluated using two real-world sequences: "Sphere" and "Car". Overall, this section provides insights into the analysis of color-based tracking algorithms on real data, highlighting the challenges and performance variations observed across different features and candidate numbers.

A. Sequence description

The sequences used for this section are described bellow.

- Sphere: consists of 200 frames capturing a closeup of a person interacting with a ball against a background that is revealed through chromatic key masking over a static image. The main challenge in tracking the ball arises from its continuously changing appearance due to the person's movements and rotations. As depicted in Figure 3, the region occluded by the hand grasping the ball varies throughout the sequence. Moreover, the image quality and color accuracy are relatively low, with a prevalent pale and brownish color tone throughout the scene. Tracking the object based solely on its color features becomes particularly difficult if the camera or compression algorithm used during sequence storage lack sufficient color information. However, it is worth noting that this scene exhibits simplicity in terms of the object's size remaining constant.
- Car: captures the rear view of a car over 399 frames using a handheld camera. Despite the car maintaining a consistent size and relative position, the camera experiences continuous shaking and movement. The background remains static, but the presence of other cars, as well as a truck located to the left of the car, introduces potential challenges in candidate selection.

B. Sequence analysis and results

For analysing this sequences several histogram features are choosen with 16 bins and the results are compared in the Table **??**. The overall performance does not vary substantially for Sphere and the Processing Time is almost constant all across the different features. It is worth mentioning how, for the car sequence, the Red channel has a higher performance since the scene is recorded during dusk so all the colors have a high blue component that makes the Blue feature less discriminatice among the cantidates.

TABLE I
PERFORMANCE COMPARISON OF DIFFERENT FEATURES FOR THE SEQUENCES SPHERE AND CAR

	Performance	
Feature	***Sphere***	***Car***
GRAY	0.672	0.637
R	0.688	**0.664**
G	0.682	0.601
B	0.651	0.613
H	**0.70**	0.505
S	0.488	0.649

To determine the optimal number of candidates for color-based tracking, we conducted a study on the "Sphere" and "Car" sequences, focusing on the HSV's Hue histogram feature for the "Sphere" sequence and the red channel feature (R) for the "Car" sequence. The results of this analysis are presented in Table II. For the "Sphere" sequence using feature H, performance values ranged from 0.687 to 0.707, with corresponding processing times ranging from 25.1066 to 45.74 milliseconds per frame. Similarly, for the "Car" sequence using feature R, performance values ranged from 0.597 to 0.666, with processing times ranging from 25.3328 to 99.329 milliseconds per frame. The results indicate that increasing the number of candidates generally improves performance, albeit with slight variations in processing time. Notably, the best performance was achieved with 121 candidates for the "Car" sequence using feature R, yielding a performance value of 0.666.

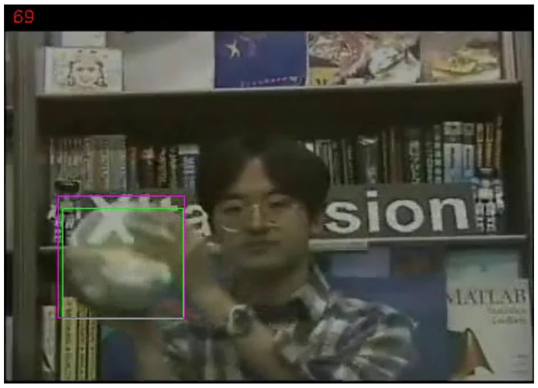

Fig. 2. Frame 69 of the sequence sphere with both gorund truth (green) and computed (pruple) bounding boxes for the Hue feature and 81 candidates (best performance).

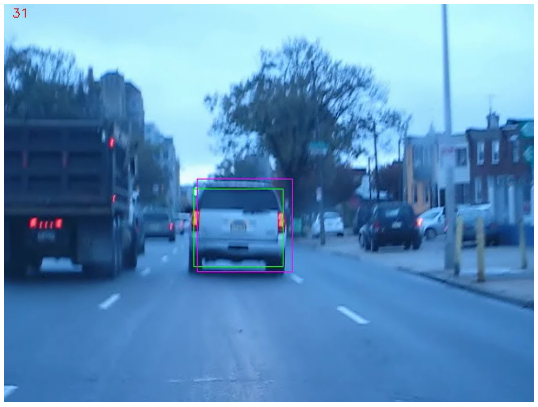

Fig. 3. Frame 31 of the sequence car with both gorund truth (green) and computed (pruple) bounding boxes for the Hue feature and 121 candidates (best performance).

V. GRADIENT-BASED TRACKING: ANALYSIS OVER REAL DATA (TASK 4.4)

In this section, the performance of gradient-based tracking algorithms is evaluated using two real-world sequences: "Sphere" and "Car". Overall, this section provides insights into the analysis of color-based tracking algorithms on real data, highlighting the challenges and performance variations observed across different features and candidate numbers.

A. Sequence description

- **Basketball**: depicts a basketball match where the primary object of interest for tracking is a player. This sequence poses significant challenges for video tracking due to the dynamic movement of the player across the

TABLE II

CANDIDATE COMPARISON FOR SPHERE AND CAR SEQUENCES.

	Shere: feature H		Car: feature R	
Number Candidates	*Performance*	*Processing Time [ms/frame]*	*Performance*	*Processing Time [ms/frame]*
25	0.687	25.1066	0.597	25.3328
49	**0.707**	45.74	0.655	44.2191
81	0.704	71.0976	0.664	70.1893
121	0.701	101.53	**0.666**	99.329
169	0.691	141.996	0.657	145.958

court and the camera's attempt to follow the player's movements. The player exhibits fast-paced actions such as running, jumping, and changing direction, making it challenging to maintain a consistent tracking on the player throughout the sequence. The camera closely follows the player, attempting to capture their movements and actions, resulting in frequent camera motion and panning. This continuous camera movement introduces additional complexities for tracking, as the object of interest undergoes significant scale changes, occlusions, and variations in appearance and orientation. Furthermore, the basketball court environment presents various elements that can interfere with the tracking process. These elements include other players, referees, spectators, and the basketball itself, which can obstruct the player or momentarily overlap with the player's region of interest. Overall, the combination of the player's dynamic movements and the camera's attempt to follow them makes this basketball match sequence particularly challenging for video tracking algorithms.

- **Ball2**: the sequence consists of 41 frames capturing a person kicking a football ball inside a goal. The main challenge in this sequence lies in the significant size difference between the ball and the frame. The football ball occupies a relatively small area, measuring only 25 x 25 pixels, while the frame resolution is 1282 x 721 pixels. The small size of the ball relative to the frame poses difficulties for accurate tracking, as the object of interest occupies a limited number of pixels, leading to potential loss of detail and reduced feature representation. Additionally, the ball's small size can make it challenging to distinguish it from other objects or background elements, further complicating the tracking process. However, several factors contribute to mitigating the tracking challenges in this sequence. Firstly, the illumination conditions are optimal, ensuring good visibility and minimal variations in lighting throughout the frames. This helps in maintaining consistent color and texture features of the ball, facilitating its tracking. Secondly, the camera used to capture the sequence is fixed, resulting in a stable background. The absence of camera motion eliminates the need to compensate for camera movements, simplifying the tracking process and allowing focus on the ball's movement. Overall, despite the small size of the ball relative to the frame, the optimal illumination conditions and fixed camera position provide favorable conditions for tracking the football ball as it is kicked inside the goal in this 41-frame sequence.

B. Sequence analysis and results

Regarding the basketball sequence, the first study is a comparison to extract the optimal number of bins. The results displayed on table III shows that the optimal number of bins for the basketball sequence is 9 bins. A further study displayed on the Table IV shows that the optimal number of candidates is 25. The performance is far from ideal and, as shown in the Figure 4, the tracking is completely of on a relatively early frame of the sequence (28 out of 500 frames).

TABLE III

PERFORMANCE COMPARISON FOR SEQUENCE BASKETBALL WITH DIFFERENT NUMBER OF HISTOGRAM BINS.

Number of bins	Processing time	Tracking Performance
4	36.186	0.051
6	32.956	0.162
8	33.212	0.062
9	33.88	0.3095
10	35.347	0.179
11	35.1146	0.05187

TABLE IV

PERFORMANCE COMPARISON FOR SEQUENCE BASKETBALL FOR 9 BINS AND DIFFRENT NUMBER OF CANDIDATES..

Number of candiates	Processing time	Tracking Performance
25	33.8843	0.309
49	97.455	0.1184
121	194.31	0.198
230	325.52	0.192

The results obtained for sequence "ball2" with a configuration of 8 candidates and 30 bins are less than optimal. The average processing time of 1351.76 ms per frame indicates a significant computational burden for each frame of the sequence. This high processing time may limit real-time tracking capabilities and hinder the practicality of the algorithm. Moreover, the average tracking performance of 0.158984 is relatively low. The tracking fails completely as soon as on the frame 6 as shown on the figure 5. This suggests that the algorithm struggles to accurately track the ball in the sequence. The low tracking performance could be attributed to several factors, such as the small size of the ball in comparison to the frame size, which can make it

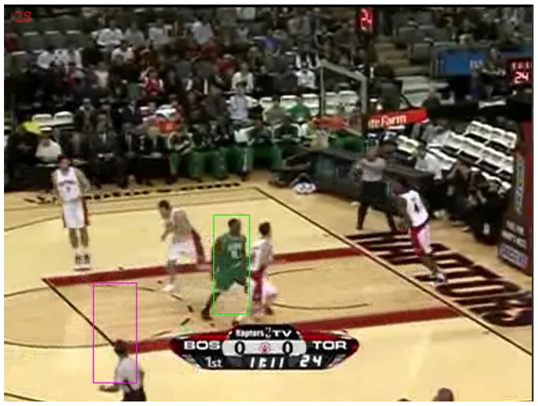

Fig. 4. Example of erroneus tracking on frame 28 of the sequence basketball with both gorund truth (green) and computed (pruple) bounding boxes for the best gradient-based tracking configuration.

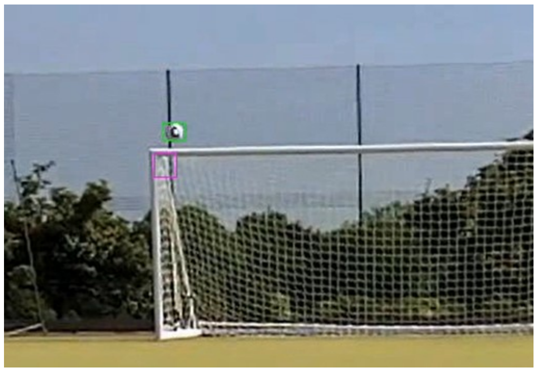

Fig. 5. Crop of Ball2 sequence on frame 6 with both gorund truth (green) and erroneus computed (pruple) bounding boxes (best performance gradeint-based configuration).

challenging to detect and track accurately. Additionally, other factors like occlusions, motion blur, or low contrast between the ball and the background might affect the tracking performance. To improve the results, several approaches can be considered. First, increasing the number of candidates might provide more robustness by considering multiple potential object locations. This can help compensate for the difficulties associated with a small object size and improve the chances of successful tracking.

The general results obtained using Histogram of Oriented Gradients (HOG) features for gradient-based tracking in the given scenario were found to be suboptimal. Several factors contribute to this suboptimal performance. Firstly, the HOG features rely on the gradient information within local image regions to capture object appearance. However, in the current sequence, the appearance of the tracked object, such as a player or a football ball, may undergo significant changes due to variations in pose, lighting conditions, or occlusions. These changes can result in inconsistent gradient patterns, making it challenging for the HOG features to effectively capture and represent the object's appearance. Secondly, the limited spatial resolution of the HOG features can also contribute to the suboptimal performance. The small size of the football ball or the player in relation to the frame resolution may lead to insufficient details in the HOG feature representation. This lack of detail can result in a loss of discriminative information necessary for accurate tracking. To address these issues and improve the performance of gradient-based tracking using HOG features, several solutions can be considered:

- Instead of simply averaging the histograms across all cells, weighted averaging techniques can be employed. Assigning different weights to individual histograms based on their spatial proximity to the tracked object or their importance in capturing discriminative features can enhance the tracking accuracy. This way, more emphasis is placed on relevant regions or distinctive characteristics of the object, while reducing the influence of less informative areas.Furthermore, incorporating spatial relationships between neighboring cells can improve the robustness of the HOG features. One option is to apply spatial pooling techniques, such as spatial pyramid pooling, which captures multi-scale information by dividing the image into different levels or regions of varying sizes. This approach enables the tracking algorithm to better handle variations in object scale, translation, and deformation, thereby enhancing the overall tracking performance.
- Incorporate additional features that capture color or texture information alongside the gradient-based features. By combining multiple feature modalities, such as HOG with color histograms or texture descriptors, a more comprehensive and robust representation of the object's appearance can be obtained. This fusion of features can enhance the discriminative power and adaptability of the tracking algorithm.
- Employ more advanced feature extraction techniques, such as deep learning-based features, may also yield better results. Deep learning models have shown significant success in various computer vision tasks, including object recognition and tracking. By leveraging pre-trained deep learning models or training custom models, it is possible to extract high-level features that capture complex patterns and variations in the object's appearance, potentially leading to improved tracking performance.

In summary, the suboptimal results obtained using HOG features for gradient-based tracking in the given scenario can be attributed to the way the histograms are extracted, to challenges in capturing the object's appearance and limited spatial resolution. Combining multiple feature modalities and exploring more advanced feature extraction techniques, such as deep learning-based features, offer promising avenues to enhance the tracking performance and overcome the limitations experiences during our experimentation.

VI. CONCLUSIONS

This laboratory report presented the implementation and analysis of a C++ program for single object tracking based on histograms using the OpenCV library. The objective of the lab was to develop a visual tracker based on color and gradient features and analyze the strengths and weaknesses of the different modules and features that compose the tracking pipeline. The report discussed the implementation details of the code, including key functions such as estimating tracking performance, creating candidates, comparing candidates, and computing histograms.

The report also described the analysis of both color-based and gradient-based tracking algorithms on real-world sequences. The sequences presented different challenges, such as changing appearance, occlusions, camera movements, and variations in color accuracy. The performance of different histogram features and the number of candidates was evaluated, providing insights into their impact on tracking accuracy and processing time.

Overall, the results demonstrated the effectiveness of histogram-based object tracking for localizing and tracking objects of interest in video sequences. The low accuracy on some of the real-world sequences highlighted the importance of selecting appropriate features and tuning parameters such as the number of bins and candidates for optimal tracking performance. The findings from this lab contribute to a better understanding of the strengths and limitations of histogram-based tracking methods and provide a foundation for future work in this area.

In conclusion, the implementation and analysis of the C++ program for single object tracking based on histograms using the OpenCV library proved to be a valuable learning experience, allowing for practical exploration of visual tracking techniques and their application to real-world scenarios.

VII. TIME LOG

The time invested for this Lab is shown in the following points:

- Starting: 0.5 hours of studying/understanding the sample code and setting up projects.
- Task 4.2: 2 hours of merging the code Extraction algorithm.
- Task 4.3: 4 hours of parameter search and problem-solving
- Task 4.4: 2 hour debugging, parameter search
- Report: 6 hours of report writing and saving results for figures.

The total time load for this second lab has been, approximately, of 14.5 hours.

APPENDIX I
COLLABORATIVE PROGRAMMING EXPERIENCE

This was our second time approaching a coding task collaboratively, and we found it to be easier compared to our previous experience. This time, we worked together in person rather than online, and we noticed that it was more efficient for us. Additionally, we were able to learn from our past mistakes and improve our approach. Collaborating in person allowed us to work more closely together and share ideas more effectively. In our specific case, we were working on a C++ project with the OpenCV library. Collaborative programming proved to be particularly useful in identifying and correcting syntax errors and compilation-time mistakes in real-time. As we learnt from our previous mistakes, we shifted our focus towards improving the quality of our individual code, which made the merging process much smoother. By ensuring that each of us had written clean and well-structured code, we encountered fewer challenges when combining our work. It became evident that investing extra effort in refining our individual parts resulted in a more seamless collaboration. This time we only spent 2 hours with merging our individual codes and cleaning it.

The structure of our collaborative coding sessions remained consistent, with each of us taking turns every 30-60 minutes to actively code and provide support to one another. Given the nature of the task, which involved extensive parameter search, it was beneficial for us to discuss and plan beforehand. We would discuss the potential challenges and difficulties we might encounter, allowing us to approach the task with a shared understanding and strategy in mind. Additionally, these discussions facilitated the exchange of ideas and perspectives, enabling us to brainstorm possible solutions together.

In conclusion, this experience of engaging in real collaborative programming for an image processing task was highly beneficial. It provided us with the opportunity to learn from our past mistakes and improve our skills. We found the experience to be valuable and are eager to continue practicing collaborative programming in future projects. We strongly encourage others to embrace this approach as well, as it fosters growth and effective teamwork.

REFERENCES

[1] S. Birchfield, ”Elliptical head tracking using intensity gradients and color histograms,” Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.98CB36231), Santa Barbara, CA, USA, 1998, pp. 232-237, doi: 10.1109/CVPR.1998.698614.