Vision-Based Assistance for the Memory Card Game

Lorenzo Ghessi
Chalmers University of Technology
Gothenburg, Sweden
ghessi@student.chalmers.se

***Abstract*—This project focuses on the development of a computer vision system designed to assist and monitor a game of the classic card game "Memory." The system's primary goal is to act as a guide for two players, keeping track of correctly matched card pairs and each player's score. To achieve this, the system implements several key image analysis functionalities. These include robust detection of the location of all cards on the playing surface, distinguishing between face-down and face-up cards, and identifying the images on face-up cards to determine if they form a matching pair. The card identification approach is based on extracting SIFT features from reference images and detected card ROIs, followed by descriptor matching and RANSAC algorithm for reliable match estimation. The system manages the game flow, providing instructions to players and updating scores based on found matches, simulating a complete game from the initial card layout to a winner declaration. Experimental evaluation demonstrates the system's ability to locate, identify, and track card states throughout various game stages.**

I. Introduction

Memory is a popular card game, especially among children, which is used to enhance and test visual memory. The game includes regulations for a deck of cards which are comprised of two packs of the same images turned face-down on a table. One player, on each turn, flips over two cards hoping to discover the same image. When the images match, the player retrieves the cards from the table, scores a point, and plays again. Otherwise, face-down again, the cards are flipped, and the other player's turn. The one who wins is the one who has the most pairs.

The objective of this project is to design a computer vision system design to act as a guide and referee for a two-player Memory game. The system is programmed to examine the game state at every turn, completing such essential functions as: identifying the location of all cards within the image, establishing the status of each card (face-up or face-down), and determining the images on the face-up cards to evaluate if they are a pair. Furthermore, the system controls the match progress by displaying the turn of the player in question, updating the scores of both players, and lastly declaring the winner.

The major approach to localizing the cards on the playing surface is blob detection and contour analysis. Following the detection of individual cards, their corresponding regions of interest (ROIs) are cropped from the original image. Subsequently, to identify face-up cards, a descriptor matching approach is employed, based on SIFT features. Descriptors are precomputed from a reference image set, with every image capturing a different type of card. Matching of the ROI descriptors against the reference ones, assisted by the RANSAC algorithm for geometric verification, enables proper card recognition.

Fig. 1: Set of cards used in this project

Generative artificial intelligence tools were used for the development of some parts of the code. Details of their use and the prompts given are recorded separately.

This report is structured as follows: the "Method" section outlines the system architecture and implemented techniques for every functionality. The "Experimental Evaluation" section presents a whole simulation of a game with intermediate results and quantitative measures to evaluate the system's performance and reliability. Finally, the "Conclusion" section summarizes results achieved and makes observations on potential future work. An appendix includes solutions to the theoretical questions asked by the project.

II. Method Section

In traditional Memory games, the number of cards can vary, but these are typically composed of pairs depicting images of different themes and arranged in an ordered grid on the table.

To simplify the development of the system, game situations involving at most 16 square cards were taken into consideration, although the system could also work with a larger number. The arrangement of the cards was assumed to be as usual, that is, in a grid, not touching and not overlapping. Furthermore, the acquired images are taken perpendicularly to the playing surface. The assumptions made imply that the cards do not present arbitrary rotations (only multiples of 90 degrees) and are arranged randomly in an image that can see the scene taken from different angles. The assumptions made are fundamental in the development of the system since in different cases, other approaches would have been necessary for the correct functioning of the program.

The development of the system was substantially divided into three steps: detection of the cards, identification of the revealed cards and of the pairs, game logic and interaction between the players.

For each step, the logic used and the functions employed will be explained. Some functions were created or provided during the lab sessions held in the course, therefore their explanations will be more limited.

A. Card Detection

The base function from which to start developing the guidance system is the detection of the face-down cards on the table. This function could have been approached using various techniques, such as edge detection, template matching, or approaches based on neural networks trained for object detection. For simplicity of implementation and subsequent development of the system, blob detection was considered the most effective strategy.

This technique is based on detecting regions of the image that exhibit homogeneous properties, such as color intensity or brightness, and that stand out from the surrounding background. In our case, the technique is effective because the cards are easily distinguishable from the surface on which they are placed.

The main operation of blob detection is divided into several key steps. First, a pre-processing of the images is performed, where the image is usually converted to grayscale and a smoothing filter, such as a Gaussian filter, is applied to reduce noise and make the regions more homogeneous. Next, the image is converted into a binary image to be subjected to thresholding. In this step, a threshold value is chosen and used to distinguish the areas in the image: pixels with an intensity above the threshold are considered part of a blob, while the others are considered background. At this point, the pixels belonging to the blobs are grouped in order to form individual regions among which, in our case, are the cards we want to identify. In fact, blobs that are not of interest may be found, so the results are filtered based on the characteristics of what we want to detect. Since we want to find square cards, limits were imposed on the areas of the blobs, the number of vertices, and the aspect ratio.

In the system, the blob detection technique was implemented through the function `find_cards_in_image_blob`, which receives as parameters the path of the photo to be analyzed and the expected number of cards, and returns a copy of the original image for drawing, the contours of the detected cards, and the original color image.

The function reads the image, which is converted to grayscale and filtered with a Gaussian filter. To apply the thresholding, the *cv2* function `adaptiveThreshold` was used, which applies the thresholding process to the previously blurred image. Compared to a simple thresholding, the adaptive thresholding function was preferred to obtain a better result. This function indeed uses an adaptive term for small regions of the image, which allows for a dynamic self-adjustment of the threshold based on the local pixel intensity, improving the detection of images in conditions of different lighting and contrast in the environment. The function is exactly what we needed since the brightness varies significantly in different parts of the image. One of the most important values of the function is c, used to set the sensitivity of the thresholding and set to 20 after various testing attempts.

To improve the image quality in order to find the contours of the possible cards, the function `cv2.morphologyEx` was used to fill holes inside the white areas found and to merge nearby white regions.

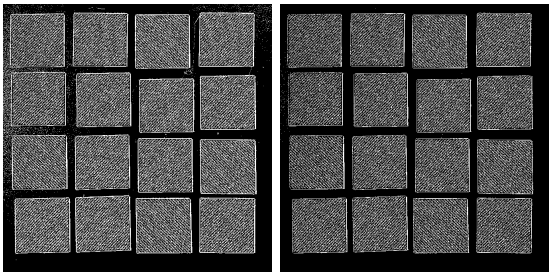

Fig. 2: Image after the thresholding with c=5 and c=20

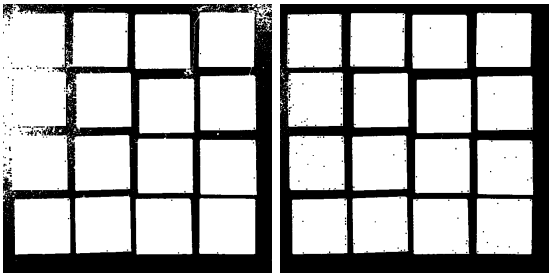

Fig. 3: Image after the morphing with c=5 and c=20

The idea at this point was to find the contours of the image and apply conditions to isolate the cards, as previously explained. To do this, given the need to apply several *cv2* functions, I was assisted by an LLM, which not only completed the function according to my requests but also helped

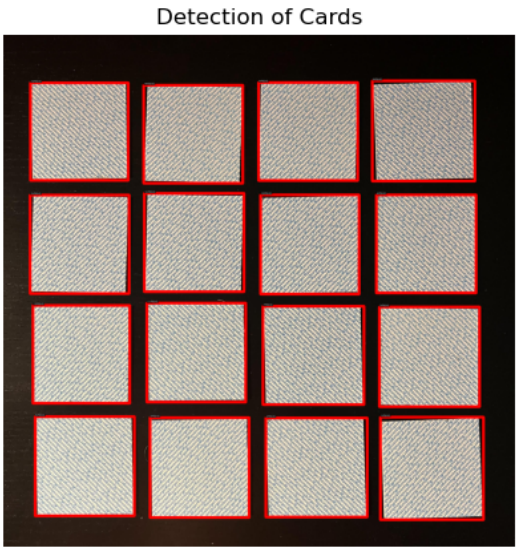

Fig. 4: Detection of the cards on the table

me finish the base functionality of the system. In fact, once the function returned the contours representing the detected cards, a for loop was used to draw red contours around the found cards and display them on the screen.

B. Card state and identification

Once the cards are localized, the next step is to determine which ones are face-up and to identify their content. For the latter task, the SIFT (Scale-Invariant Feature Transform) algorithm was employed. SIFT is an algorithm used to detect and describe the keypoints of an image, building descriptors based on the orientation and intensity of local gradients. The descriptors are then used for comparison and recognition of points between different images. The applied idea consists of extracting SIFT descriptors and keypoints from reference images of each card type in the deck. These references are then compared with the features extracted from the regions of interest (ROI) of the cards detected in the game image via blob detection. If a ROI matches a reference with sufficient confidence, the card is identified; otherwise, it is assumed to be face-down.

1) Creation of the reference database SIFT: To build the feature database of the reference cards, the function `load_sift_references` was used, in which the functions `augment_data` and `extract_sift_features`, already used in previous labs, are employed. This function receives as input parameters the path to the dataset containing the reference images of each card type, the desired size for resizing `target_size` of these images, and a data augmentation factor `augmentation_m`. The latter is used by the `augment_data` function, indicating the number of random rotations to apply to the reference image to better learn the descriptors and make recognition more robust. For the implementation of the logic to iterate through the dataset subfolders (each corresponding to a card type) and for the management of the data structure storing the keypoints and descriptors associated with each card ID, assistance from an LLM was requested. The result of this function is a dictionary that acts as a SIFT database, mapping the ID of each card type to a list of (keypoints, descriptors) pairs obtained from its reference images (original and augmented).

2) Game scene analysis and ROI identification: After loading the SIFT database and a game scene with some face-up cards, the new image is analyzed to find the cards on the table using the function `load_and_detect_cards` (essentially the same as `find_cards_in_image_blob` but simpler to read). Subsequently, the scene is analyzed to extract the ROIs necessary for the comparison. For each ROI, a state is updated containing an index, the card state understood as face-up or face-down, the name, the coordinates of the card, and the inlier count set to zero. Then, through the function `identify_single_card`, the previously extracted ROI is compared with the SIFT database. Inside `identify_single_card`, for the ROI under examination, its SIFT features (keypoints and descriptors) are first extracted using `extract_sift_features`. Subsequently, its descriptors are compared with those of each reference image present in the SIFT database using the function `match_descriptors`. If a sufficient number of matches is found (greater than `MIN_RAW_MATCHES_BEFORE_RANSAC`), the corresponding keypoints are used to estimate an affine transformation via the RANSAC algorithm (`ransac_fit_affine`). This process helps filter out incorrect matches and provides a robust measure of similarity. If a match is found, the state, ID, and inlier count of the ROI are updated. It is important to note that a threshold `MIN_INLIER_COUNT_FOR_ID` was used to determine the number of inliers necessary to determine the ROI state. Here too, functions already seen previously were used to implement this function: `match_descriptors` for comparing the SIFT descriptors of a ROI with the ones of each card in the database, `ransac_fit_affine` to compute a robust affine transformation between the pairs of corresponding keypoints, `residual_lgths` to quantify the number of inlier, and `extract_sift_features` for extracting the keypoints and the descriptors.

3) Pairs recognizing and visualization: Finally, once all ROIs in the image have been analyzed, the function `find_pairs` was created to find the pairs of cards, simply by analyzing the ID of the two recognized face-up cards. Then the function `draw_and_show_board` is used to draw red boxes in case if face-down cards, green boxes in case of pairs of face-up cards, or yellow boxes for non-pair face-up cards.

III. Experimental evaluation section

A. Game implementation

The game logic has been implemented following the indications provided in the project description. The match begins by showing the 16 covered cards and the detection of them. For each turn, the active player turns over two cards, and the

system identifies if the cards are a pair. If the player has found a pair the system signals it with the message *"Congratulations! A matching pair. Please pick up your matching card pair and then you may continue"*, signalling the new score and giving the possibility to the player to continue. Otherwise, the system's output is *"No match! Turn back the two cards face down again"*. Player X may then continue, and the turn passes to the other player. The match goes on until the 8 pairs are not found, at that point the system declares a winner or in the case of a draw. The simulation of the entire match is taken from a subfolder in which all the rounds are contained in a proper order. For managing the extraction of the images from the subfolder, a function created via LLM has been used.

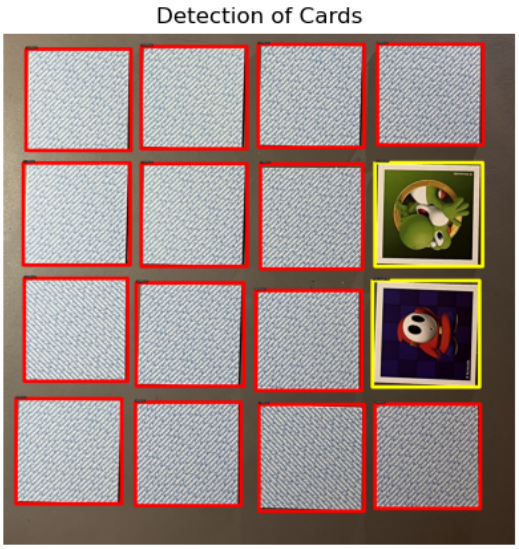

Fig. 5: Detection of different cards

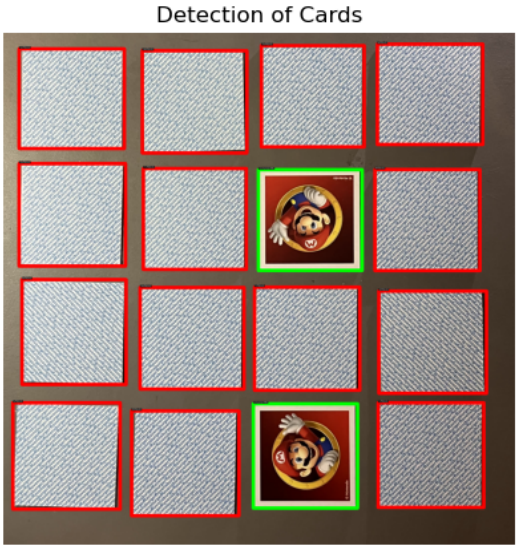

Fig. 6: Detection of a pair of cards

B. Performance metrics

During the game simulation, the system performed as expected, always providing the correct outputs based on the analyzed game situation. To more thoroughly assess the system's performance, 40 images different from those used during the game simulation were provided. By analyzing these images, it was possible to compute quantitative metrics for the system functions. The analysis was carried out by printing the 40 images after they had been processed. A counter kept track of the total number of detected cards, covered cards, and uncovered cards. By visually analyzing the output on the images, the correctness of the results was evaluated. The code used to print the images was created using an LLM, with the prompt provided in the appropriate section.

1) Card detection: In the 40 images provided to the system, the number of cards present was 504, all of which were detected by the system. This indicates excellent robustness in detecting the cards present on the game table.

2) Face up/down detection: To evaluate the system's performance in detecting whether a card was face-up or face-down, a confusion matrix was built. The values obtained are shown below:

TABLE I: Confusion Matrix

	Positive	**Negative**
Positive	TP = 78	FN = 2
Negative	FP = 1	TN = 423

Where the different values indicate:

- TP: the system predicted a face-up card and it was actually face-up.
- FN: the system predicted a face-down card but it was actually face-up.
- FP: the system predicted a face-up card but it was actually face-down.
- TN: the system predicted a face-down card and it was actually face-down.

From these values, the following parameters can be calculated:

$$Accuracy = \frac{TP + TN}{tot} = 99.4\%$$

Indicates the overall percentage of correct classifications.

$$Precision = \frac{TP}{TP + FP} = 98.7\%$$

Indicates how often the system correctly predicted that the cards were face-up.

$$Recall = \frac{TP}{TP + FP} = 97.5\%$$

Indicates, among all face-up cards, how many were detected by the system.

3) Card identification: When the system detects a face-up card, the percentage of correctly identifying the card—and thus the pair—is 100%. This is because the method for identifying a card is the same as that for recognizing a card as face-up, i.e., comparing the ROI in the image with the available SIFT database. If the system identifies the card as face-up, it means it has recognized what card it is.

Despite the excellent performance of our system, it is important to mention that these results are achieved in cases where the cards are on dark, solid-colored, and uniform surfaces, in addition to the assumptions made initially. In cases of different surfaces such as those shown in Figure 7, where the background is light, or in Figure 8, where the background is not uniform, the system completely loses its effectiveness because it fails to detect the cards. In these cases, blob detection fails: an approach based on edge detection or neural networks could be more effective.

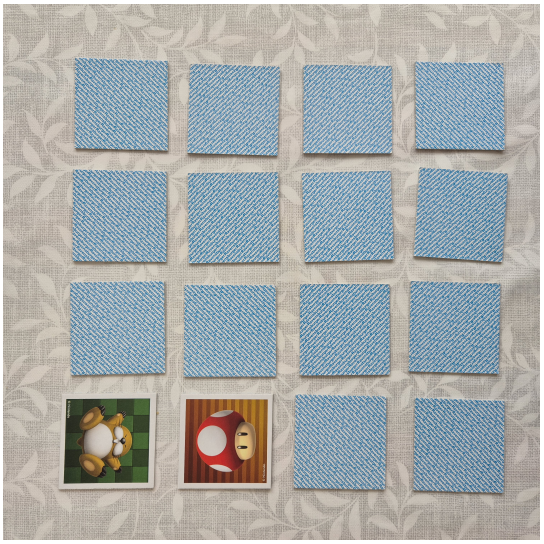

Fig. 7: White background

IV. Conclusion

The project resulted in an excellent implementation of a vision system to assist two players in a memory game. After a simple pre-processing of the images, the SIFT technique and the RANSAC algorithm were used to detect the face-up cards and the present pairs. While the system is highly accurate in the tested scenarios, it remains limited when using lighter backgrounds or brighter environments. A possible improvement could be to strengthen the system to make it reliable under different conditions as well. Additionally, a further step could be to remove certain assumptions, such as the grid layout of the cards or the perpendicularity of the image to the playing surface. The ability to operate under these conditions would lead to a significant enhancement of the system, making it an ideal game assistant for any situation.

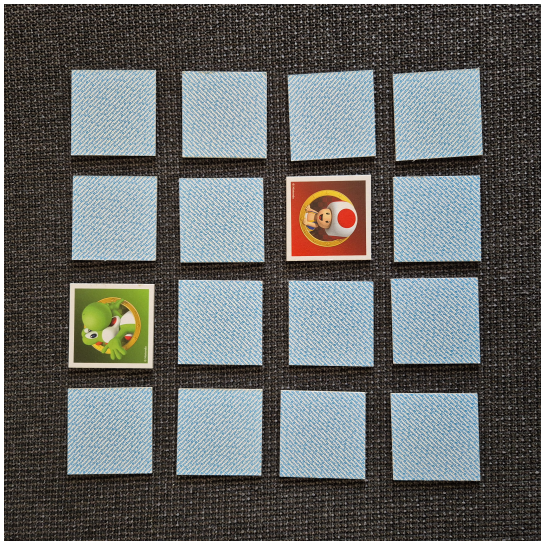

Fig. 8: Not uniform background

V. Theoretical Part

A. Overfitting

1) Explanation: Overfitting is an undesirable behavior that occurs in machine learning when a model provides excellent predictions for the training data provided but not for the new data input to the system. The phenomenon usually occurs when a model learns ”too much” (hence the name overfitting), capturing both the data we would actually want it to learn and noise, fluctuations, or specific features of the dataset provided. A very common example used to better understand overfitting involves the analysis of dog images. In this case, the model used must analyze a set of photos to identify in which of them dogs are present. If the model has been trained with many images of dogs on grass, the model might take the grass as a feature for classification and, as a consequence, might not identify the dog in another circumstance. Another example that can be provided is represented by the medical diagnosis of symptoms. Considering a model that learns to associate certain symptoms with specific diseases, if the dataset provided is not varied enough (for example, it comes only from patients from one hospital), the system might specialize in the specific dataset given but end up being inaccurate when new data to analyze is input. From these two examples we understand how, with overfitting, the model learns the noise and coincidences of the training set more than the underlying patterns, leading to inaccuracies and malfunctions.

2) Causes: The main causes of the overfitting are four:

- Dataset too small or poorly representative: the dataset might be not sufficient, not containing the necessary data to represent accurately all possible input data, thus learning characteristics present in the limited set and leading to imprecision in the analysis of other samples.
- Presence of noise in the training data: the dataset might contain large quantities of not very relevant information,

termed noisy data, that the model might learn, reducing its own ability to generalize.
- Prolonged training: the model is trained for too long on the same dataset, leading to memorizing the training data as irrelevant details and noise.
- Model too complex: given the complexity of the model, its learning capacity is greater, leading also in this case to learning the noise present in the dataset in addition to the significant patterns present.

3) Solutions: Two solutions usually used to avoid overfitting in the context of neural networks are:

- Dropout: it is a regularization technique in which, during training, for each batch of data a random fraction of neurons in multiple layers of the network are temporarily turned off. By randomly eliminating neurons, this technique reduces the dependence between the neural layers, forcing each neuron to learn more robust features autonomously. In addition, dropout leads the network to be simplified differently at each iteration, introducing a noise that avoids an excessive specialization on the training data.
- Regularization: it is a technique in which a penalty term on the model's complexity is introduced, that is, we alter the model's loss function by adding a regularization parameter so that the model does not become too complex and therefore, less prone to capturing the noise.

B. Generative models

1) Optimization of ELBO rather than $log(p(x))$: The Variational Autoencoder is a probabilistic generative model that learns to represent complex data by compressing them into a latent space and reconstructing them from it. The VAE is composed of an encoder, which takes an input x and maps it into a distribution of latent variables $q(z|x)$, usually a Gaussian $\mathcal{N}(\mu, \sigma^2)$, the latent space, in which the variable z is present and represents the compressed information, and a decoder, which takes z and reconstructs $p(z|x)$ trying to make it resemble x. The objective to reach when training the model is to maximize the likelihood of the observed data x, therefore to maximize the function $log(p(x))$. The problem is that to compute it, it would be necessary to integrate over all the possible configurations of the latent variable z, that is, to compute:

$$log(p(x)) = log \int p(x, z)dz \tag{1}$$

Since the computation is too high, what is done is to approximate the distribution $p(z|x)$ with $q(z|x)$ and optimize the ELBO, defined as the lower bound of the likelihood function.

$$ELBO = E_{z\sim x}[\log p(x|z)] - D_{KL}(q(z|x)||p(z)) \tag{2}$$

Where the first term indicates how well the decoder can reconstruct x given z, and the second is called Kullback-Leibler divergence, which penalizes the differences between the learned distribution $q(z|x)$ and the one assumed before observing the data $p(z)$.

2) Differences between VAEs and Diffusion probabilistic models: Diffusion probabilistic models are generative models that are based on two processes: a forward one in which noise is added to the data in order to obtain a distribution that is usually Gaussian, and a reverse one in which the diffusion process is inverted to generate new data. From the working principle, it can be understood that they operate in a different way compared to VAEs. Going into detail, the differences are:

- In the encoding process, in the VAE, there is a single pass through the neural network in which the input x is directly mapped to the parameters of the probability distribution in the latent space z. In diffusion models, there is no direct mapping to a latent space but there is the forward diffusion process, which is iterative, and gradually adds noise to the input x.
- In the VAE, the latent variable z has lower dimensions than the input x, so that the most important features of the data are captured. With training, the goal is then to bring the distribution of the encoded z closer to the prior distribution. In diffusion models, instead, the latent variables have the same dimension as the input, to which a quantity of noise is added at each step.
- In the VAE, decoding is done in a single pass, taking a point z from the latent space and transforming it into a new sample in the data space x. In diffusion models, instead, the generation process, called reverse diffusion, is done iteratively by removing step by step the noise added previously. This process is not learned, which is why diffusion models are slower than VAEs, even though they produce higher quality results.

C. Triangulation

1) How to obtain the 3D point that best represents the 2D measurements: With triangulation, the goal is to determine a 3D point X using its 2D projections (x_i) in k images, given by the projection matrices P_i. To do this, a two-step approach is used where a nonlinear refinement to find the solution is performed after obtaining a first linear solution. This is because the 2D measurements may be affected by noise and, consequently, the projection of the 3D point may not be accurate, so a subsequent nonlinear refinement is performed to minimize the error.

- To find the linear solution we must take the fundamental relationship that links the 3D point X and its image projection $x = (x, y, 1)$, both in homogeneous coordinates, through the projection matrix P:

$$\lambda x = PX \tag{3}$$

In the equation, λ is a scalar representing the depth. The equation can be rewritten as:

$$\lambda \begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} X \tag{4}$$

Since \$w=1\$, we can derive a system of equations from which we remove λ, by inserting the third row into the first and second.

$$(P_3 x - P_1)X = 0 \tag{5}$$

$$(P_3 y - P_2)X = 0 \tag{6}$$

The system can then be solved for X using a minimal solver such as SVD.

- A better estimate is then obtained by refining the position of the 3D point X, minimizing the sum of the squared reprojection errors across all images. The objective is therefore to minimize the cost function:

$$\min \sum_{i=1}^{k} \|x_i - \hat{x}_i\|^2 \tag{7}$$

Where x_i is the observed image point and $\hat{x}_i = (\frac{P_1^i X}{P_3^i X}, \frac{P_2^i X}{P_3^i X})^T$ is the projection of X using the matrix P_i. To minimize the cost function, iterative techniques such as gradient descent or Gauss-Newton are used, which start from the previously found linear solution to improve the result.

2) Possible derivation of the position of x_1 and x_2: Considering a setup with $k = 2$ images and known relative pose, there exists a situation that can lead to having infinite points that project exactly onto x_1 in the first image and onto x_2 in the second, that is when the 3D point lies on the baseline connecting the optical centers of the two cameras. This is because, in this case, a 3D point X, its projection x_1 on camera 1, and the center C_1 are collinear, i.e., they lie on the same line, just as X, x_2, and C_2 do. Introducing the term epipole, which is the projection of a camera center onto the image plane of the other, we have that if the 3D point lies on the baseline C_1C_2, the projection x_1 in image 1 will be e_1, and similarly x_2 will be e_2 in image 2. In this situation, the projective rays from the centers through the respective projections $x_i = e_i$ correspond to the baseline itself. This means that the two rays overlap, and there are therefore infinite 3D points on the line that project to x_1 and x_2. An example could be the case in which the cameras are parallel and placed facing each other: in this situation there are infinite 3D points lying on the line between the two camera centers, thus finding a unique solution for the point X that lies on it is impossible. However, as explained before, the situation can occur in all possible configurations between the two cameras. In fact, there is always a line that connects their centers and, if the point lies on it, finding a unique solution for those points is impossible.

LLMs PROMPT

Gemini Pro 2.5 has been used for implementing some parts of the code. All the times that I used it, i took the response and I changed it to better fit in my original code. Here there are the prompts that I used as mentioned during the various explanations:

- I am creating a blob detection function to recognize square playing cards, arranged on the table forming a 4x4 grid. The image is parallel to the plane, and the cards are on a dark background. The code I have written so far is used to find contours in a binarized image where the cards are white and the background is black. Finish the code so that the function returns the original image and the cards found. I will use the function to display an image with red squares over the detected cards.
- I am creating a function to extract keypoints and descriptors from each image contained in subfolders, using SIFT. I already have functions available for data augmentation and for extracting SIFT features. You can find the mentioned functions in the attached file.
- Create a function that counts the number of images in a subfolder and assigns a number to each of them. Return a two-column array where the first column contains the number and the second the filename.
- Given a folder from which to take test images, I want to apply the implemented functions to all the images and display them in rows of five elements. Create the code for me.

REFERENCES

[1] Course slides: Lectures 3, 4, 10, 11, 12
[2] Course lecture notes by Olof Enqvist
[3] https://aws.amazon.com/what-is/overfitting/
[4] https://fangdahan.medium.com/derivation-of-elbo-in-vae-25ad7991fdf7
[5] https://datascience.stackexchange.com/questions/47906/understanding-elbo-learning-dynamics-for-vae
[6] https://datascience.stackexchange.com/questions/52352/why-maximize-elbo-in-the-variational-autoencoder
[7] https://medium.com/swlh/all-about-opencv-built-in-functions-45f2db75afc5
[8] https://www.analyticsvidhya.com/blog/2019/03/opencv-functions-computer-vision-python/
[9] https://www.geeksforgeeks.org/opencv-python-tutorial/