DeepArUco++: improved detection of square fiducial markers in challenging lighting conditions

Rafael Berral-Soler [a,*], Rafael Muñoz-Salinas [a,b,**], Rafael Medina-Carnicer [a,b,**], Manuel J. Marín-Jiménez [a,b,**]

a Dpt. Computing and Numerical Analysis, University of Córdoba, Spain
b Maimonides Institute for Biomedical Research of Córdoba (IMIBIC), Spain

Abstract

Fiducial markers are a computer vision tool used for object pose estimation and detection. These markers are highly useful in fields such as industry, medicine and logistics. However, optimal lighting conditions are not always available, and other factors such as blur or sensor noise can affect image quality. Classical computer vision techniques that precisely locate and decode fiducial markers often fail under difficult illumination conditions (e.g. extreme variations of lighting within the same frame). Hence, we propose DeepArUco++, a deep learning-based framework that leverages the robustness of Convolutional Neural Networks to perform marker detection and decoding in challenging lighting conditions. The framework is based on a pipeline using different Neural Network models at each step, namely marker detection, corner refinement and marker decoding. Additionally, we propose a simple method for generating synthetic data for training the different models that compose the proposed pipeline, and we present a second, real-life dataset of ArUco markers in challenging lighting conditions used to evaluate our system. The developed method outperforms other state-of-the-art methods in such tasks and remains competitive even when testing on the datasets used to develop those methods. Code available in GitHub: `https://github.com/AVAuco/deeparuco/`

Keywords: Fiducial Markers, Deep Neural Networks, Marker Detection, CNNs

1. Introduction

Fiducial square planar markers are artificial images used in computer vision applications to facilitate tasks such as camera and object pose estimation, object detection and tracking, and navigation. These markers encode data that allows

---

*Corresponding author

**Contributing authors

Email addresses: `rberral@uco.es` (Rafael Berral-Soler), `rmsalinas@uco.es` (Rafael Muñoz-Salinas), `rmedina@uco.es` (Rafael Medina-Carnicer), `mjmarin@uco.es` (Manuel J. Marín-Jiménez)

for precise identification within a scene and have a regular shape that helps to determine their orientation relative to the camera. Examples of such markers are ArUco [1] (see Fig. 2), AprilTag [2] or ARTag [3], which are utilized on fields such as healthcare [4, 5] and robotics [6, 7].

Fiducial markers can also be used as a source of information for 3D reconstruction [8] and mapping techniques like SLAM [9] (Simultaneous Localization and Mapping). These methods rely on locating reference points (dubbed keypoints) to keep track of an agent's position within a scene. However, finding such keypoints can be challenging, especially in regions that lack texture information. In these cases, fiducial markers can complement existing keypoints or even replace them entirely when valid keypoints cannot be found.

While classic fiducial marker detection and identification methods perform well in controlled settings, they are not robust under unexpected conditions, as they require adequate lighting or careful fine-tuning to perform at their best [10]. For instance, classic ArUco relies on image contour thresholding and polygon extraction to detect markers in an input image [11, 12]. In this case, blur can make borders challenging to detect, and lighting conditions can render thresholding settings useless.

On the other hand, convolutional neural networks (CNNs) have seen great success in tasks such as image object detection and recognition, being more robust and flexible than classic computer vision techniques. Since the success of AlexNet [13], deep CNNs have retained their state-of-the-art status in such tasks without additional restrictions. These methods can improve robustness towards blur and lighting through the use of data augmentation techniques [14].

Inspired by this, we propose a novel method (Fig. 1) using deep CNNs to perform the detection, precise location, and decoding of square fiducial planar markers. While we target ArUco markers, the method should readily adapt to other square planar markers, provided they encode information in a similar way. We needed a large dataset of ArUco markers in varying orientations and lighting conditions to develop our methods. As we did not find any publicly available datasets meeting our requirements, we developed a method for creating a synthetic dataset that can be used to train our models, providing a high degree of flexibility in the generation of samples. Additionally, we captured a real-life dataset of ArUco markers in challenging lighting conditions to test our approach and provide a fair comparison with preexisting methods.

The contributions of this work are the following:

- First, we present a new framework composed of three separate models: marker detector, corner regressor, and marker decoder, based on deep CNNs. These models are combined in a pipeline to perform detection, precise location, and decoding of ArUco markers. Using disjointed models at each step allows for a more straightforward approach to the problem and favors modularity. This method outperforms classic ArUco and DeepTag [10] in challenging lighting conditions while keeping a competitive throughput. Also, the presented method stays competitive compared to other datasets (namely the DeepTag dataset).
- Second, we introduce a simple but effective method for generating syn-
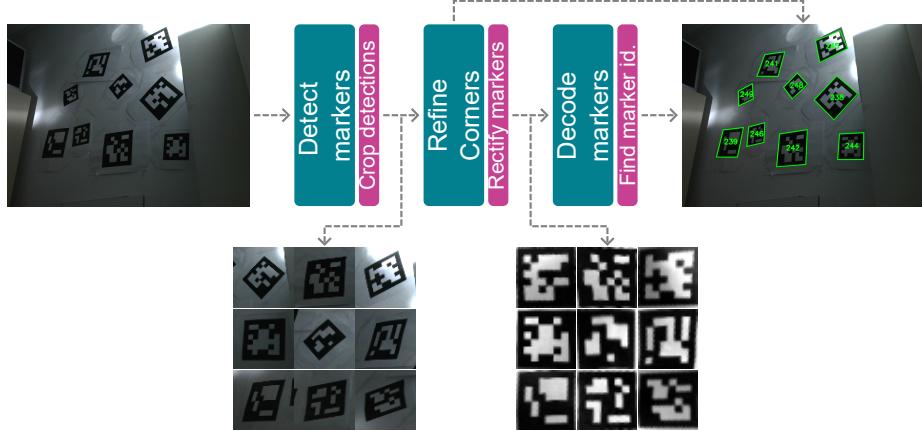
2

Figure 1: **Main: DeepArUco++ framework.** The marker detector receives as input a color image and returns a series of bounding boxes, which are used to obtain crops from the input image. Then, the corner regressor model is applied over each crop to refine the position of the corners. The detected markers are then rectified with the refined corners and used as input for the marker decoder. Finally, the marker is assigned the ID with the least Hamming distance w.r.t. the decoded bits. (Best viewed in digital format)

thetic datasets containing ArUco markers, used for the training of each model in the pipeline.

- Third, we captured a second, real-life dataset of ArUco markers in challenging lighting conditions for testing our methods. Finally, the implementation of our methods and the datasets we produced will be published online for public use (see the URL provided in the Abstract).

While our method incorporates knowledge from previous works in object and keypoint detection, our proposed combination, trained on a specially composed synthetic dataset designed to replicate challenging lighting and shadow effects, results in highly robust marker detection even under harsh lighting conditions. To the best of our knowledge, this is the first work that aims to achieve precise marker detection and decoding under such difficult lighting conditions.

An earlier version of this work was presented at IbPRIA'23 [15]. The main improvements in this new work are the following: first, we aim to improve the detection capabilities of our method and explore a new method for marker corner location based on the use of heatmaps; additionally, we compare our models against other state-of-the-art techniques both on our test dataset and on the DeepTag dataset, from a performance standpoint and from the perspective of the throughput of our models.

The rest of the paper is organized as follows. First, Section 2 summarises some related work. The datasets created for developing and testing our methods are introduced in Section 3. Then, we describe our proposed approach in Section 4. Section 5 presents our experiments and discusses their results. Finally, we conclude the paper in Section 6.
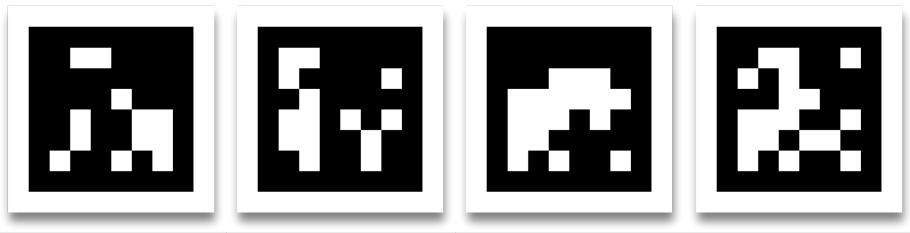
Figure 2: **Some ArUco marker examples.** The type of fiducial marker used in this work. The bits can be read from left to right, top to bottom (black means 0, white means 1), discarding the outer white and black border (the two outermost "rings") to produce a sequence of 36 bits (6×6), that is compared against the ArUco dictionary to obtain the encoded ID. (Best viewed in digital format)

2. Related Works

Square planar fiducial markers [12] are a popular computer vision tool used whenever precise camera pose estimation is required, used in tasks such as robot navigation and object tracking in augmented reality. ArUco [16], AprilTag [17], and ARTag [3] are some of the most popular ones, used in both industry and academia; a previous implementation of ArUco [1, 18] is available for its use in the OpenCV computer vision library.

These markers usually follow a similar structure: they are shaped as a square, with black borders that mark the boundaries of a grid of black and white square cells, each representing a different bit value. Information can be encoded in every marker using those bits, allowing for the precise identification of individual markers in the scene.

2.1. Classical techniques.

Fiducial markers are usually detected and decoded through classical computer-vision techniques. These techniques assume adequate lighting and the absence of additional factors impacting image quality and can fail outside controlled conditions. Some works based on classical techniques aim to perform in suboptimal conditions. Authors in [19] address the problem of fiducial marker identification under different adverse conditions, such as blur and non-uniform lighting. However, they do not aim to precisely locate the markers in the scene. On the other hand, authors in [11] locate markers in images achieving resilience towards blurriness, but they do not address the challenge of inadequate lighting. To our knowledge, the task of jointly locating and decoding fiducial markers under challenging lighting conditions remains unaddressed.

Despite the importance of lighting for developing robust object detectors, there is limited literature on object detection under poor lighting settings. Without specialized hardware, such as IR cameras, the most straightforward approach would be to apply a classic image-enhancing technique to achieve better contrast and then use an off-the-shelf object detector over the corrected images. Some of those enhancing techniques are based on thresholding (binary,

4

Otsu [20]), gamma correction or histogram equalization. Although straightforward, these techniques often depend on carefully selected values (e.g., gamma or threshold values) and may not account for extreme lighting variations between regions of the same image. Adaptive histogram equalization techniques, like the ones described in [21] do adjust to different regions of the image, but can lead to amplified noise on homogeneous regions, making object detection even more challenging.

2.2. Deep Neural Networks.

Methods based on deep learning, such as the ones described in [22] or [23], learn the correspondence between low-lighting images and their normal lighting counterparts, leading to better results than those obtained through the use of classical techniques. However, using a neural network as a preprocessing step is computationally intensive and may lead to longer processing times. This could be mitigated by end-to-end training of a single detector model, starting from in-the-wild images taken under varying lighting conditions or using data augmentation to simulate them, making the image enhancement step unnecessary.

In the past few years, there have been some attempts to tackle the problem of joint marker detection and identification using Convolutional Neural Networks. In [24], a method for detecting boards of markers under low-light and high-blur conditions is proposed. However, this method is designed around a fixed configuration of markers on a board (referred to as ChArUco) and cannot be used to locate and decode individual markers. Authors in [25] leverage YOLOv3 to detect ArUco markers under occlusions. However, this work focuses only on marker detection, not corner refinement or marker identification; lighting conditions are also ignored. Finally, a new work has been published [10], using a MobileNet backbone to detect markers and iteratively refine their corners. However, their work focuses mainly on sensor noise and motion blur; thus, it does not address the problem presented in this work.

Our proposed approach addresses the problem of detecting markers in challenging lighting conditions in the following manner. First, we developed a new synthetic dataset featuring markers under varying lighting conditions. While mild blur has been used as a data augmentation technique, this is done to obtain a more robust model (i.e. our work does not explicitly tackle blur conditions). Next, we trained a pipeline consisting of an off-the-shelf marker detector along with custom-designed corner refinement and marker decoding models using this dataset. While recently sophisticated methods based on transformers [26, 27, 28, 29] have achieved significant success in general-purpose object detection tasks, we have chosen to use YOLOv8 [30] as our object detector due to its efficiency and ease of training and use on mid-range computers. The resulting system can detect and correctly identify ArUco markers under challenging lighting settings, outperforming other state-of-the-art methods (namely ArUco [16] and DeepTag [10]) in this task.

3. Datasets

We have used the following datasets to develop and test our methods and compare them against preexisting ones.

3.1. Flying-ArUco v2 dataset

To train our methods, we have developed a synthetic dataset, dubbed Flying-ArUco v2 dataset, composed of images containing various ArUco markers overlaid on backgrounds sampled from the MS COCO [31] 2017 training dataset.

To create our dataset, we initially sampled 2500 images from the entire MS COCO 2017 training dataset. These images have been rotated to get a wide aspect ratio and cropped to get valid backgrounds of size 640×360, discarding images smaller than this target size. We sampled our backgrounds using the following approach: first, we calculated the median *luma* [32] (or *lightness* component) for each image by converting it to grayscale using the `cv2.COLOR_BGR2GRAY`[1] conversion. This conversion is needed to obtain a single brightness value per pixel, as using the original RGB image (with three values per pixel) would not yield a meaningful result after the median calculation (we would just be computing the most common intensity value for *any* channel). Next, we sorted the images into 10 equal-sized bins, meaning each bin contained the same number of images, although the range of brightness values differed across bins. This binning method allowed us to match the brightness distribution of the source dataset. Finally, we sampled an equal number of backgrounds from each bin to ensure a balanced representation. Overall, the luma range for our samples extends from 0.0039 to 0.9608, on a scale of 0 to 1.

Up to 20 markers sampled from the ArUco DICT_6X6_250 dictionary (composed from 250 ArUco markers with 6×6 bits, for more information please refer to the OpenCV ArUco documentation[2]), with varying sizes, orientations, and camera focal distances, have been overlaid on each resulting background. Fake markers have also been included to provide negative examples: full-black markers, markers with inverted colors, markers containing colors different from black or white, and other markers containing patterns composed of random lines and Perlin noise instead of the expected grid of black and white square cells.

Finally, to get a more natural result on the synthetic images, markers have been combined with the background by projecting the luma of the background image over the overlaid markers; doing so makes the lighting of the overlaid markers more consistent with the lighting of the background, making it more challenging to find the added markers just by picking the items with odd lighting. As the background images present a wide variety of lighting settings, the resulting images contain markers under a myriad of lighting and shadow combinations. Please note that this dataset is intended to be used as a tool in the

---

[1]Color conversions on OpenCV: `https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html`

[2]ArUco on OpenCV: `https://docs.opencv.org/4.x/de/d67/group__objdetect__aruco.html`

Figure 3: **Flying-ArUco v2 dataset.** Using images from the COCO dataset as background, challenging training samples are created by overlapping markers with varying poses, sizes, and positions in the image. Top: detection dataset. Bottom: refinement/decoding crops. (Best viewed in digital format)

development of marker detection methods, and not as a way to assess model performance. Examples of the resulting images are depicted in Fig. 3.

To train our detector, we have split the dataset into two subsets, one for train and the other for validation. The training set contains 2000 images, and the validation set contains 500 images. The markers included in each subset are cropped to obtain the corner regression and marker decoding datasets: the markers from the training subset will be used as the training dataset, and the markers from the validation subset will be used as the validation dataset.

3.2. Shadow-ArUco dataset

To test the methods we have developed and compare them against other methods, we have obtained a second dataset in real-world conditions, dubbed the Shadow-ArUco dataset. This dataset was initially presented in IbPRIA'23 [15].

For the construction of this dataset, we have attached multiple markers to the walls in the corner of a darkened room (we closed the blinds and turned off the lights); then, a video containing moving shadows and lighting patterns is projected over the corner. The entire scene is then recorded from different points of view, obtaining multiple video sequences.

To estimate the ground-truth labels of the markers in the scene, we have used a semi-manual method: first, the frames are processed using the traditional ArUco method, setting the error correction bits (ECB) parameter equal to zero to get only the detections with the highest confidence, as this weeds out most false detections. Since the camera is fixed during each video sequence, marker positions are consistent across frames. If a marker is not detected in a particular frame, its corners and ID can be retrieved from the nearest frame where it was detected. As a last resort, if a marker is not detected, we manually add the position of its corners and ID and then copy this information for every frame in the sequence.

We have recorded the scene from six different points of view, obtaining six different video sequences for a total of 8652 frames, with a resolution of 800×600. Examples of the images contained in this dataset are depicted in Fig. 4.

7

Figure 4: **Shadow-ArUco dataset.** A video containing complex lighting patterns is projected on the corner of a room, over whose walls multiple patterns have been attached; the complex lighting makes marker detection difficult for classical methods. The scene is recorded from multiple fixed camera positions. (Best viewed in digital format)

Figure 5: **DeepTag ArUco dataset.** A single ArUco marker is photographed in a fixed environment at varying distances and orientations w.r.t. the camera. (Best viewed in digital format)

3.3. DeepTag ArUco dataset

Finally, to provide a fair comparison against the model proposed in [10], we have tested our methods on the DeepTag dataset built by the authors of the method. The dataset can be obtained at the DeepTag project page [3]. This dataset comprises photographs of different fiducial markers in fixed positions at varying distances and orientations with respect to the camera. For each position, a total of 100 photos are obtained. While the authors do this to take into account sensor jitter, images obtained for a given marker type from a particular position do not have significant differences; nevertheless, we will consider each independent frame as a different image for testing purposes. Some examples from this dataset appear in Fig. 5.

As the original dataset contains examples for multiple marker families, we restrict our comparison only to those images containing ArUco markers. Taking this into account, the resulting dataset contains markers photographed with view angles of $\{0, 10, 20, 30, 40, 50, 60, 65, 70, 75\}$ degrees at distances of

---

[3]DeepTag project page: `https://herohuyongtao.github.io/research/publications/deep-tag/`

$\{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$ cm., for a total of 100 different positions with respect to the camera, for a total of 10000 frames (100 frames per position), with a resolution of 1280×960. We will restrict our comparison to the `general` dataset (i.e., we will not measure the performance of our methods in the images containing Gaussian noise or simulated motion blur, as these tasks fall out of the scope of this work). Please note that each image contains a single marker and is the same for every image (i.e., the encoded ID is the same).

4. Proposed DeepArUco++ Framework

Our approach to marker detection and identification is built around a pipeline obtained by dividing the problem into simpler tasks. The initial task involves detecting the markers and their bounding boxes (see Section 4.1). This is done without finding the ID of the marker; i.e., the overall position of the marker is found without decoding it. Then, for every detected marker, the precise location of its corners is determined (see Section 4.2): this allows us to determine the exact location of the marker to rectify it, making decoding it easier. Finally, the markers are decoded by determining the value of each bit and comparing the extracted bits against the ArUco codes dictionary (see Section 4.3).

4.1. Bounding-box level detection

The first step in our pipeline is the broad-brush location of the markers contained in the frame. This is done using a model based on the YOLOv8 object detector; we have considered both its nano (`yolov8n`) and small (`yolov8s`) sized variants, starting from the provided pre-trained models. These models were trained using the Flying-ArUco v2 dataset, presented in Section 3.1.

While we have started from off-the-shelf YOLOv8 models, improvements regarding robustness towards challenging lighting conditions stem from the use of the dataset designed in Section 3.1 as training data. As a form of offline data augmentation, we have considered multiple approaches. The most general, applied for every augmented frame, consists of multiplying each image by a synthetic gradient with an arbitrary angle, with random minimum and maximum values ranging from 0.0 to 2.0; this is done to simulate an even broader range of lighting conditions. Additionally, other transformations have been considered, such as shifting the color of every frame by a small amount (i.e., applying a particular "tint" to the input image), applying full-frame Gaussian blur (with an overall probability of 20%) or adding a varying amount of Gaussian noise to the frame. In total, nine additional augmented samples are created for each non-augmented sample (the size of the dataset grows by a factor of $10\times$). Different transformations and model size combinations have been used to find the best-performing configuration when testing on both Shadow-ArUco and Deep-Tag ArUco datasets. No online data augmentation has been used, and other parameters, such as the loss function, have been kept as default.
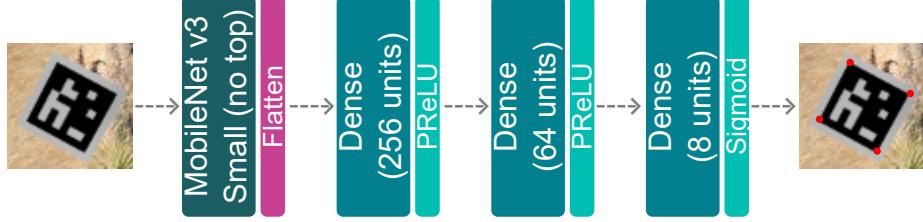
9

Figure 6: **Corner regressor architecture.** The model's input is a 64×64 color image containing a marker; the output is a vector containing eight values corresponding to the x and y coordinates of the four corners of the marker in counterclockwise order. Best viewed in digital format.

4.2. Corner regression

The second step in our pipeline is the precise location of the corners of each marker to better fit the actual shape and dimensions of the detected marker. Two alternative approaches have been considered for this task: (a) direct corner regression (i.e. directly regressing the coordinates of each corner) and (b) heatmap-based corner regression (i.e. directly locating the corners in the input image, to later obtain the precise coordinates). Please note that as presented in this work the two methods are mutually exclusive: only one of them can be used at a time, as they provide the same functionality in our proposed pipeline.

4.2.1. Direct corner regression

The first approach we have considered for the task of corner regression is that of directly regressing the coordinates of each corner, i.e., predicting the eight different values corresponding to the (x, y) coordinates of each corner of a given marker (two values for each corner). We have used a model based around a pre-trained MobileNetV3 backbone (in its `MobileNetV3Small` variant). We removed the original, fully-connected top from this backbone and kept the rest as a feature extractor. Then, we added a custom top composed of two fully-connected layers with sizes 256 and 64, with a PReLU activation function, and a third fully-connected layer, with size 8 (two coordinates for each corner) and a sigmoid activation function as the output. The network expects a 64×64 pixel color image as input and outputs four pairs of coordinates within the $[0, 1]$ range, representing the positions of the corners relative to the width/height of the input image. A summary of the architecture appears in Fig. 6.

To train the models, the markers from the Flying-ArUco v2 dataset (see Section 3.1) are cropped by using their ground-truth bounding boxes (previously expanded by a margin of 20% over the original width at both left and right and a margin of 20% of the original height at both top and bottom), and then resizing to a size of 64×64. Markers from the training subset (after offline data augmentation) will be used to train the model, and markers from the validation subset will be used to verify the progress of the training. As a form of online data augmentation, images will be rotated by a multiple of 90°, mirrored horizontally
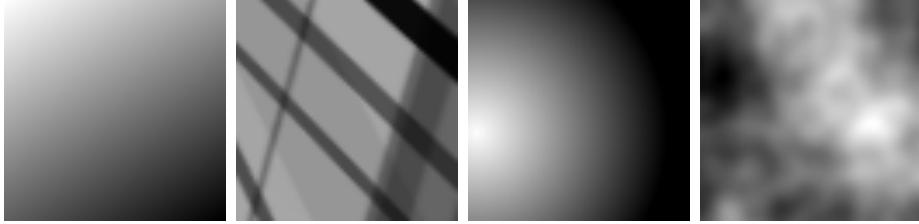
Figure 7: **Lighting patterns used for data augmentation.** From left to right: simple gradients, lines, circular patterns, and Perlin noise. Multiple patterns can be combined (i.e., multiplied) to achieve even more complex augmentations.

or vertically, and multiplied by a lighting pattern composed of lines, Perlin noise, and/or circular shadows (see Fig. 7); multiple transformations can be applied to the same sample. Labels associated with each input must also be transformed for consistency with the transformations applied to the crops (i.e., the coordinates of each corner and their order must reflect the rotations and mirroring applied to the markers). As the loss function, we will use the mean average error (MAE) between the predicted coordinates and the (properly transformed) ground-truth corners' positions.

In inference time, the input to the model will be obtained by expanding the bounding boxes found by the marker detector as we did to obtain the training crops. Then, markers are cropped and resized to a size of 64×64, and the resulting image is used as the input to the corner regressor. Final positions are computed by multiplying each output by the width/height of the bounding box as corresponding, then offsetting each position by the minimum x and y values of the expanded bounding box.

4.2.2. Heatmap-based corner regression

As a second approach to precise corner location, we have implemented a heatmap-based approach. Taking inspiration from keypoint-based methods, such as those used in human pose estimation and facial landmarks location [33, 34], we compute for each coordinate a Gaussian heatmap, reflecting a probability density function, following a normal distribution whose mean corresponds to the position of the encoded corner. Then, the maps corresponding to each corner are combined to obtain a single map reflecting the most probable locations of every corner in the image. As the architecture for this model, we have used a U-Net [35], as similar approaches have been used with relative success in keypoint detection tasks [36, 37]; marker corners can be considered as a type of keypoint. A summary of the architecture used in this approach appears in Fig. 8.

As a heatmap built in such a way is mainly composed of values near zero, two corrections must be made to allow our models to learn from these images. The first is to try and maximize the number of non-zero values in the heatmap: to do so, we have increased the values in the covariance matrix by a factor of ten.
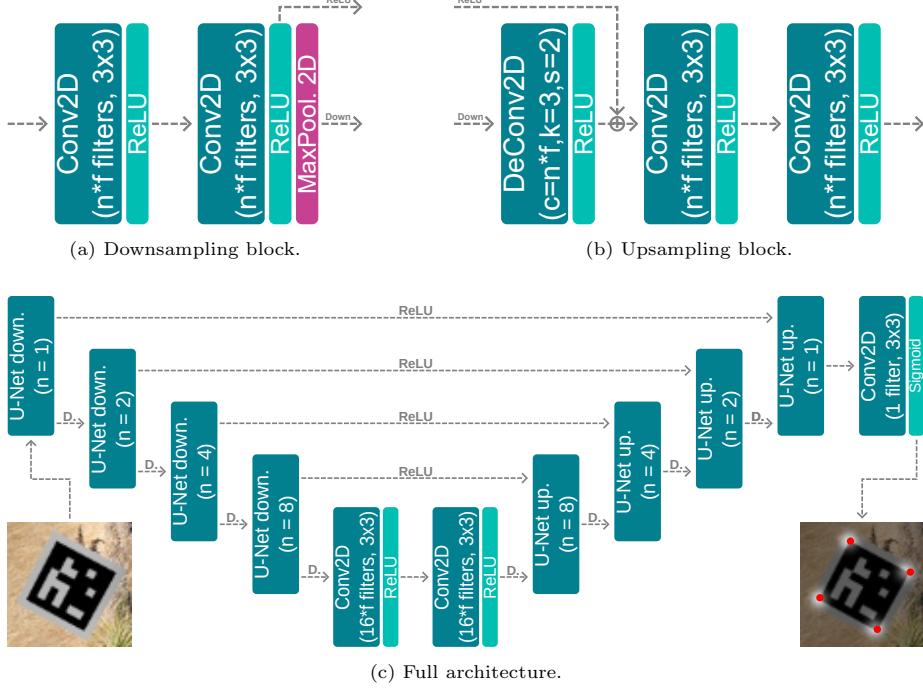
(a) Downsampling block.

(b) Upsampling block.

(c) Full architecture.

Figure 8: **Heatmap-based corner regressor architecture.** The model's input is a 64×64 color image containing a marker; the output is a heatmap containing *blobs* in the areas corresponding to the detected corners. In strided convolutions (`DeConv2D`), c stands for the number of filters, k stands for the kernel size ($k \times k$), and s stands for the used stride size ($s \times s$). In (b), $+$ signals a concatenation of inputs. In (c), $D.$ stands for *Down* (as in (a) and (b), abbreviated for readability). Best viewed in digital format.

Additionally, we have used a weighted loss function based on the mean squared error (MSE), weighted in such a way that the error near the corners we aim to predict is valued up to ten times as much as the error far from the corners. See Equations 1 and 2 for a better understanding of the loss calculation, where Y stands for the target heatmap to predict, r and c stands for the number of rows and columns in the output, respectively, i stands for the row number (from 0 to $r-1$), j stands for the column number (from 0 to $c-1$) and $\mathcal{L}_h$ stands for the weighted loss function.

$$weights = \frac{Y - Y_{min}}{Y_{max} - Y_{min}} \cdot 9 + 1 \tag{1}$$

$$\mathcal{L}_h(Y, \hat{Y}) = \frac{1}{r \cdot c} \sum_{i}^{r} \sum_{j}^{c} (\hat{Y}_{i,j} - Y_{i,j})^2 \cdot weights_{i,j} \tag{2}$$

To train the models, we have used the markers from the Flying-ArUco v2 dataset (see Section 3.1) cropped and augmented as described in Section 4.2.1,
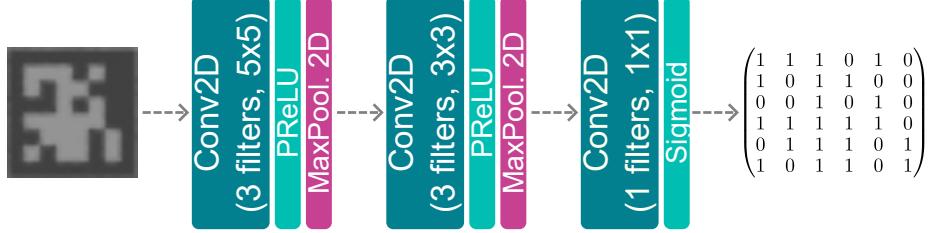
Figure 9: **Marker decoder architecture.** The model's input is a 32×32 grayscale image containing a rectified marker; the output is a matrix of size 6×6, containing the prediction for each bit in the input picture, in the range [0-1].

but using the heatmaps computed from the ground-truth corners as the labels to predict. These markers are also normalized to the $[0, 1]$ range.

On inference time, the input for the models will be obtained the same way as for the direct regression approach. However, to use the outputs, we have to compute the corner positions from the regressed heatmaps. To do so, we propose the following method: first, we extract the *blobs* (areas with high values relative to the rest of the map) from the regressed map using the blob extractor provided by the OpenCV library, tuned to the expected blob area (i.e., finding blobs with an area similar to that of the blobs in the training heatmaps). The average value is computed for every detected blob and used as its score. The top four blobs are used to calculate the refined corner positions (please note that it is possible for the method to locate less than four corners). From every blob, a mask is computed (to discard every value outside the detected blob). The final position (x, y) is computed as the sum of the coordinates of each non-zero valued position, weighted by the contribution of the value in that position to the sum of all values in the blob. For a better understanding of the position calculation from a given blob, see Eq. 3), where *blob* stands for the heatmap after zero masking every pixel not belonging to the blob of interest and i and j indicate a row and a column in the heatmap, respectively.

$$x, \; y = \sum_{i,j} j \cdot \frac{blob_{i,j}}{\sum_{i,j} blob_{i,j}}, \; \sum_{i,j} i \cdot \frac{blob_{i,j}}{\sum_{i,j} blob_{i,j}} \tag{3}$$

4.3. Marker decoding

The last step in our pipeline is extracting the bits encoded in each marker. We will use a simple architecture composed of three convolutional layers (see Fig. 9). As input, the model expects a 32×32 pixels grayscale image, normalized with respect to its maximum and minimum values to the $[0, 1]$ range. As its output, it returns a 6×6 array of values in the range $[0, 1]$, which, after rounding, can be interpreted as the decoded bits.

We have used the markers from the Flying-ArUco v2 dataset to train the model, cropped from their ground-truth corner positions. As a form of online

data augmentation, we shift ground-truth coordinates by up to 5% in each direction before cropping the markers; this is done to simulate "imperfect" corner detections to obtain a more robust model; additionally, we have used the same lighting, flipping, and rotation transformations as for the training of the corner refinement models (see Section 4.2). We used the MAE between the predicted values and the ground-truth encoded bits as a loss function.

On inference time, the input for the model is obtained using the corners predicted by the corner refinement model in the previous step; then, after locating the four corners, the marker is rectified through a homography, by finding the transformation which maps each corner to its corresponding corner in a completely flat square. After rectifying, the values in the resulting image are normalized to the $[0, 1]$ range and used as input for the decoder network. Then, the output of the decoder is flattened and rounded to the nearest integer (i.e., 0 or 1), and the resulting sequence of bits is compared against every possible code in the ArUco dictionary. As there are four possible orientations for the marker, the comparison is performed four times, rotating $90°$ each time (i.e., the output 6×6 matrix is rotated, flattened, and finally compared). The marker is then assigned the ID corresponding to the ArUco code with the smallest Hamming distance with respect to the encoded bits in any orientation. Finally, by defining a maximum Hamming distance as a threshold, we can discard false detections based on the extracted ID; however, this will likely translate into a smaller recall (i.e., some correctly detected markers may be discarded).

5. Experiments and Results

5.1. Metrics and methodology

To validate our methods, we have used a combination of multiple metrics, testing each stage of the pipeline individually. To measure the capabilities of the different marker detectors obtained in this work, we have used a Precision-Recall curve, starting from the detections and ground-truth marker locations in the form of bounding boxes. We consider a detection correct if it overlaps with some ground-truth bounding box with an *Intersection-over-Union* (IoU) score of at least 0.5. The models will be tested over the Shadow-ArUco dataset described in Section 3.2.

To test the performance of our corner refinement models, we will measure the average per-corner error, computed as the distance in pixels between a predicted corner and its nearest ground-truth corner, and the time spent per frame, measured in milliseconds. Additionally, for the methods based on heatmaps, we will get the percentage of markers for which one, two, three, or four corners have been found (as the described method may find less than four corners). As input for the models, we will use the ground-truth marker annotations from the Shadow-ArUco dataset, cropped and expanded as described in Section 4.2.1.

Marker decoding was assessed using a different Precision-Recall curve, each point representing the precision and recall values obtained after filtering the output of the decoder using a different threshold. For this test, recall is the

ratio between the number of detected markers remaining after discarding those for which the closest ArUco code is found at a distance greater than the target threshold and the total number of ground-truth markers. Precision will be computed as the ratio between the number of correctly identified markers at a distance below that target threshold and the number of markers remaining after the previous filtering.

Finally, we will test the capabilities of the entire pipeline at different configurations and compare them against previous state-of-the-art fiducial marker detection methods (classic ArUco and DeepTag). We will also consider the previous iteration of this work in [15] as a baseline. In this comparison, we will use Precision-Recall curves to test the detection capabilities of each method. DeepArUco++ and baseline DeepArUco will have their detections filtered by thresholding at the decoding step. Additionally, a numerical, quantitative comparison will be provided, measuring the area under the Precision-Recall curve (AUC-PR) for every method, the minimum precision and maximum recall values (useful to test the detection capabilities of the different methods under no additional filtering), the average per-corner error and the percentage of accuracy, computed as the number of correctly identified markers with respect to the number of true detections (i.e., detected markers which correspond with a real, ground truth marker) after filtering. A comparison of the time spent per frame with each method will be provided, with a per-step breakdown for the different DeepArUco++ configurations.

5.2. Implementation details

We have used Ultralytics YOLOv8 [30] to train our marker detector. Both our corner regressor and marker decoder have been implemented using Tensor-Flow. Our experiments were run on a computer with an Intel(R) Core(TM) i7-11700F CPU and an NVIDIA GeForce RTX 3090(R) GPU.

We trained the marker detector using rectangular training (see Ultralytics YOLO documentation[4]), with an IoU threshold of 0.5 for non-maximum suppression, automatic batch size computation (it varies between executions, based on the available amount of GPU memory), and a maximum of 1000 epochs, with early stopping after 10 epochs without validation improvement. The rest of the parameters used to control the training of the detector were set to default; a learning rate of $1e-2$ and automatic selection of optimizer (using SGD in our runs, with a momentum of 0.9). For the corner refinement models, we used the Adam optimizer with a batch size of 32, a maximum of 1000 epochs, early stopping after 10 epochs without improvement, and learning rate halving after 5 epochs without improvement (initial learning rate: $1e-3$). Refer to Section 4.2 for loss functions. The marker decoder was also trained with the Adam optimizer, batch size of 32, and up to 1000 epochs, with early stopping after 20 epochs and learning rate halving after 10 epochs without improvement (initial learning rate: $1e-3$). The loss function used was the mean average error

---

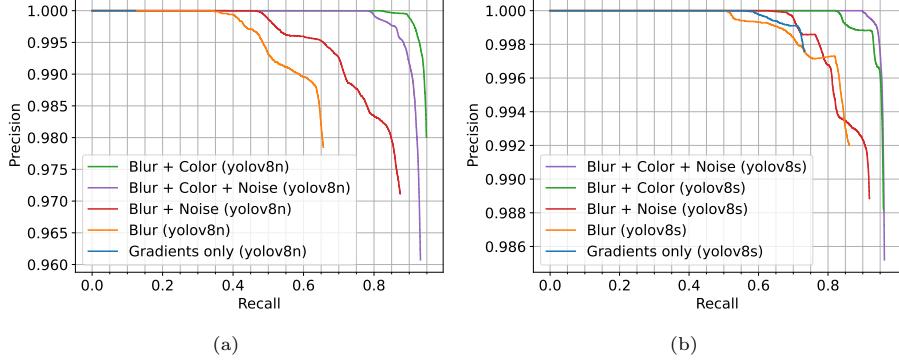[4]YOLOv8 documentation: `https://docs.ultralytics.com/usage/cfg/#train-settings`

Figure 10: **Comparison of different training configurations on Shadow-ArUco dataset.** **(a)** Precision-Recall curve for models based on `yolov8n` (nano variant), **(b)** Precision-Recall curve for models based on `yolov8s` (small variant). Legend reflects the options used for data augmentation in each training; *Color* stands for color shift; *Noise* stands for Gaussian noise. Best viewed in digital format.

(MAE). We refer the reader to the public source code of DeepArUco++ (see Abstract) for further implementation details.

5.3. Method development

5.3.1. Marker detection

From the methodology described in Section 4.1, we have trained 10 different marker detectors using different combinations of augmentation options and base model sizes. In Fig. 10, we compare the Precision-Recall (PR) curves obtained when testing over the Shadow-ArUco dataset from Section 3.2.

We can extract the following conclusions from the comparison in Fig. 10. From a Precision-Recall standpoint, the optimal approach appears to be training a model from a pre-trained `yolov8s` model, using blur, color shift and Gaussian noise to produce augmented samples (see Section 4.1), with an AUC of 0.9622. However, if we consider the method's efficiency, using a model based on `yolov8n` will likely result in a better throughput. Training from a pre-trained `yolov8n`, and using both blur and color shift as augmentation options, the resulting model has an AUC of 0.9485, not far from the best `yolov8s`-based model. Overall, it seems that the best combination of options when considering both model sizes is to use just blur and color shift, as it yields the best performance when training from `yolov8n`, and gets the second place when training from `yolov8s` (with 0.9592, versus 0.9622 when using every option).

5.3.2. Corner regression

Following the methodology described in Section 4.2, we have trained five different corner refinement models; one based on the MobileNetV3-based architecture from Section 4.2.1, which performs the direct regression of the coordinates of the four corners, and the rest based on the U-Net based architecture from
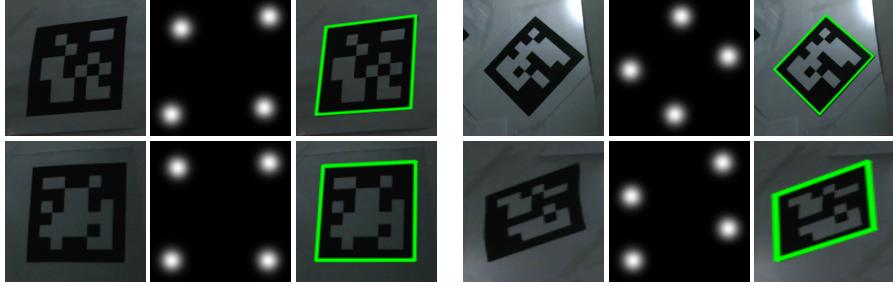
16

Figure 11: **Examples of heatmaps and marker locations obtained using our method.**
For each block, the left row shows the original crops, the middle row shows the obtained
heatmaps and the right row shows the obtained precise marker location. Best viewed in
digital format.

Section 4.2.2, which locates the corners in the form of a heatmap (five differ-
ent configurations have been tested, based on the number of filters on the first
convolutional layer). The models have been trained on the cropped markers ob-
tained from the Flying-ArUco v2 dataset, cropping after adding blur and color
shift to the whole image, as it yielded the best overall results for the training of
the detectors on Section 5.3.1 (online data augmentation has also been applied
on a crop-by-crop basis, following the method on Section 4.2.1). Some examples
of heatmaps generated using our method are shown in Fig. 11.

Please note that in our design for the heatmap-based refinement model, it
is possible to locate less than four corners; also, for both the heatmap-based
approach and the direct regression approach, we discard detections with their
nearest ground-truth corner found at a distance greater than 5 pixels. A com-
parison of the results obtained by the different models, testing over the crops
obtained from the Shadow-ArUco dataset (using the expanded ground-truth
bounding boxes), appears in Table 1.

From the results in Table 1, we observe that the heatmap-based approach
clearly improves the corner detection rate. Additionally, it improves the aver-
age per-corner error (mostly reducing the variability), especially with $f = 32$.
On the other hand, it has a much higher inference time than the direct re-
gression approach. This is caused by the need to extract the actual corner
positions from the heatmap; much optimization is required on the implementa-
tion side. Considering this, we recommend using the heatmap-based approach
when maximum performance is required, leaving the direct-regression approach
for scenarios where throughput is a concern.

5.3.3. Marker decoding

Using the method described in Section 4.3, we have trained a simple marker
decoding model based on the architecture from Fig. 9. The models have been
trained on the cropped markers obtained from the Flying-ArUco v2 dataset,
augmented by adding blur and color shift as we did for the corner refinement
models in Section 5.3.2. In Fig. 12, we can see the Precision-Recall curve ob-

Table 1: **Comparison of different corner refinement models on Shadow-ArUco dataset.** *Error* stands for the mean Euclidean distance between a predicted corner and its nearest ground-truth corner (in pixels); *Time* refers to the mean inference time per frame, in milliseconds (with 10.90 ± 1.33 markers per frame); f refers to the number of filters in the first convolutional layer (when applicable). Percentages of markers for which a certain number of corners have been found have been computed over the total number of markers in the Shadow-ArUco dataset. Note that *Error* only takes into account detected corners (could be less than 4 corners). For metrics noted with ↓, lower is better.

Method	# of found corners					Error ↓	Time ↓
	0	**1**	**2**	**3**	**4**		
Direct	0.74%	2.43%	5.70%	10.97%	80.16%	1.55 ± 1.01	$\mathbf{3.29 \pm 3.69}$
Heatmap $(f = 2)$	0.01%	0.23%	2.33%	9.50%	87.93%	1.61 ± 0.85	6.24 ± 3.45
Heatmap $(f = 4)$	0.00%	0.07%	0.79%	9.65%	89.48%	1.51 ± 0.77	6.34 ± 3.59
Heatmap $(f = 8)$	0.00%	0.06%	0.89%	8.63%	90.41%	1.51 ± 0.72	6.44 ± 3.82
Heatmap $(f = 16)$	0.00%	0.06%	0.49%	8.68%	90.77%	1.51 ± 0.71	6.57 ± 4.50
Heatmap $(f = 32)$	0.00%	0.04%	0.52%	8.44%	**91.00%**	$\mathbf{1.44 \pm 0.69}$	6.87 ± 4.63

tained for the trained model on the markers from the Shadow-ArUco dataset (cropped and rectified from their ground-truth corner positions) at different maximum Hamming distances. From the curve, we can see how, even using a threshold of 9, it is possible to correctly identify most markers (with a precision of 0.9903); however, using a maximum distance of 3, we can still achieve a recall of 0.9476 while keeping a perfect precision.

A relevant question regarding our pipeline is whether improvements in corner detection lead to improvements in marker identification or not, for instance, due to limitations in the approach to marker decoding. In Fig. 13, we can see a comparison of the corner refinement precision of each method in Table 1 in relation to the number of correctly identified markers obtained when applying the marker decoding model to the crops refined through such method. For each method, starting from the ground-truth marker locations (i.e., not using the detection methods described in Section 4.1), we crop and refine every marker in the Shadow-ArUco dataset from Section 3.2, and we obtain the number of markers for which the IoU with respect to the ground-truth counterpart is greater or equal than two different thresholds, namely 0.5 and 0.9; additionally, we apply
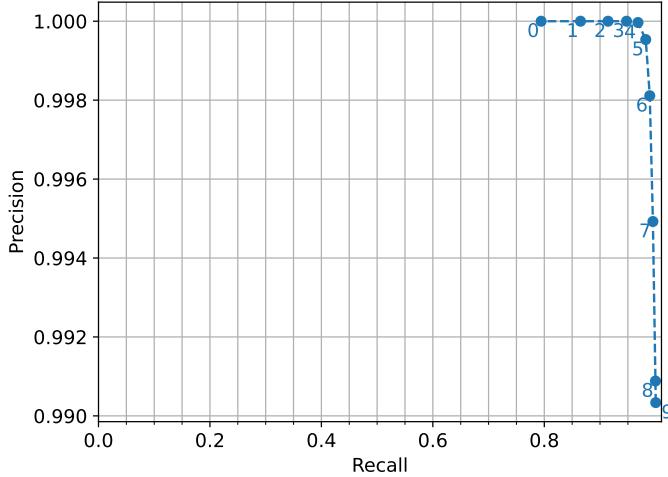
Figure 12: **Precision-Recall curve for the DeepArUco++ decoder on Shadow-ArUco dataset.** We can see the precision and recall values for different Hamming distance thresholds. *Recall* is defined here as the fraction of markers with an ID found at a distance equal to or below the threshold; *Precision* is defined as the fraction of markers with a correctly assigned ID w.r.t. the total of markers after filtering by that threshold.

the marker decoding model to the markers refined by each method (without additional filtering, i.e., no minimum IoU requirement has been enforced), and count the number of correctly decoded markers (i.e. the assigned label is the same as the ground-truth label). For both metrics, we have taken into account only markers with 4 detected corners.

From the results in Fig. 13, it seems that the improvements achieved through the use of the heatmap-based approach do translate well to the number of correctly identified markers, with a near-perfect correlation (especially when considering the 0.5 threshold). This suggests that the marker detection method does not hinder improvements in the rest of the pipeline. Additionally, we can use the obtained curves to determine which method to use at the corner refinement step: for instance, while the method which correctly identified the most markers is the heatmap-based with $f = 32$, the difference between it and the one with $f = 8$ is not that significant, with 84871 and 84312 respectively. However, the difference in inference time is somewhat noticeable (see Table 1).

5.4. Ablation study.

Finally, we will perform an ablation study comparing multiple configurations of the DeepArUco++ framework against each other. As our pipeline is composed of three clearly separated steps, and every step is required for providing the function of the complete system, our ablation study will focus on the multiple proposed combinations for marker detector and corner refinement models; the marker decoding model will be the same for every DeepArUco++ configuration. Also, we will compare against two currently state-of-the-art methods
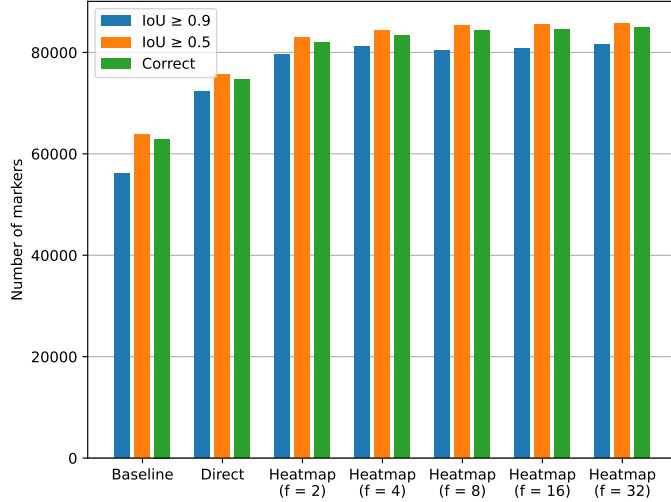
19

Figure 13: **Corner refinement precision vs. correctly decoded markers.** For each of the corner refinement models in Table 1, we compute the number of markers for which the IoU w.r.t. their ground-truth counterpart is greater or equal to each different threshold (0.5 and 0.9, respectively), and the number of overall correctly decoded markers (without additional filtering).

for the detection and decoding of fiducial markers, namely classic ArUco and DeepTag, and the baseline version of our method (DeepArUco), as presented in our previous work [15]. While the baseline DeepArUco also uses a YOLO-based detector model, it was trained on a previous version of the Flying-ArUco dataset; also, it used a larger variant (`yolov8m`, instead of `yolov8s` or `yolov8n`). The tested DeepArUco++ configurations are the following:

- `yolov8n` + direct:

  - Detector: Trained from `yolov8n`, using blur and color shift as offline data augmentation options.
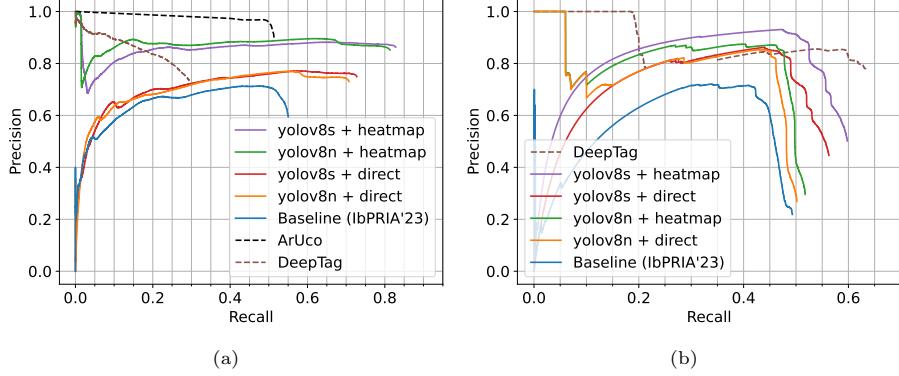
  - Refinement model: Direct regressor.

- `yolov8n` + heatmap:

  - Detector: Trained from `yolov8n`, using blur and color shift as offline data augmentation options.

  - Refinement model: Heatmap-based, with $f = 8$.

- `yolov8s` + direct:

  - Detector: Trained from `yolov8s`, using blur and color shift as offline data augmentation options.

  - Refinement model: Direct regressor.

Figure 14: **Comparison of DeepArUco++ detection capabilities against the state-of-the-art. (a)** Precision-Recall curve over the Shadow-ArUco dataset; **(b)** Precision-Recall over the DeepTag dataset. Legend reflects the considered method/setting. Best viewed in digital format.

- `yolov8s` + heatmap:

  - Detector: Trained from `yolov8s`, using blur and color shift as offline data augmentation options.
  - Refinement model: Heatmap-based, with $f = 8$.

No thresholding has been applied at the decoding step, and any markers for which less than four corners have been detected (with a maximum distance of 5 pixels to the closest ground-truth corner) have been removed. The Precision-Recall curve for each configuration over the Shadow-ArUco dataset appears in Fig. 14a.

As shown in Fig. 14a, there is a noticeable difference between the heatmap-based corner refinement model and the direct regression approach, with the former yielding significantly higher precision and recall values. For a given corner refinement model, the difference between using the `yolov8n` or the `yolov8s` detector is not that much. In any case, the detection capabilities of every tested DeepArUco++ configuration are far beyond those of ArUco, DeepTag, or even the baseline version of DeepArUco. However, it should be noted that in terms of precision, the classic ArUco is unmatched at the cost of a much-reduced recall. A numerical comparison of the different methods, considering the AUC value for the curves in Fig. 14a, the mean error at the corners and the percentage of detected markers at different levels of restrictiveness appears in Table 2.

From the comparison on Table 2, we can conclude the following: if we only consider the Intersection-over-Union (IoU) between the detected markers and the ground-truth markers (considering every predicted corner as valid, and only considering markers with 4 detected corners) the percentage of found markers (with respect to the complete Shadow-ArUco dataset) is similar for every DeepArUco++ configuration. This seems to indicate that in this dataset, the penalty when using the `yolov8n`-based detector versus the `yolov8s`-based one

21

Table 2: **Quantitative comparison of DeepArUco++ detection capabilities against the state-of-the-art, on Shadow-ArUco dataset.** *Matched B.B.* stands for the percentage of markers detected, discarding markers with less than 4 detected corners at the output of the corner refinement step; *Corners filtered* stands for the percentage of detected markers that remain after discarding every corner for which the distance to the closest ground-truth corner is higher than 5 pixels, and keeping only markers with 4 remaining corners; *Corners + ID.* stands for the percentage of markers after filtering corners for which the predicted ID. is correct; *Only ID.* stands for the percentage of detected markers for which a correct ID. has been predicted, but less than 4 corners remain after filtering; *Corner error* stands for the mean Euclidean distance between a predicted corner and its nearest ground-truth corner (in pixels). For metrics noted with ↓, lower is better; for ↑, higher is better.

Method	AUC ↑	% of ground-truth markers				Corner error ↓
		Matched B.B. ↑	**Corners filtered ↑**	**Corners + ID. ↑**	**Only ID.**	
yolov8n + direct	0.4930	94.80%	70.74%	69.64%	18.56%	1.94 ± 0.95
yolov8n + heatmap	**0.7120**	94.24%	81.34%	79.68%	8.75%	2.11 ± 0.83
yolov8s + direct	0.5107	**95.82%**	72.75%	71.38%	17.70%	1.89 ± 0.95
yolov8s + heatmap	0.7094	95.11%	**82.83%**	**80.90%**	8.26%	2.04 ± 0.83
Baseline (IbPRIA'23)	0.3530	82.35%	55.06%	54.78%	21.99%	1.77 ± 1.10
ArUco	0.5047	53.25%	51.41%	43.60%	1.40%	**0.72 ± 0.51**
Deeptag	0.2537	39.77%	29.38%	29.31%	10.13%	1.40 ± 0.93

is not that much. However, when discarding corners further than 5 pixels from their closest ground-truth corner (and keeping only markers retaining the four corners), differences arise between the heatmap-based and the direct corner regression approach, with the former resulting in a much higher yield. For every method (with the exception of ArUco), it seems that almost every marker with 4 corners found can be correctly decoded. Also, it is worth noticing that in the case of DeepTag, almost every detected marker can be decoded (counting every marker with 4 detected corners before filtering and a correctly assigned ID., for both "Corners + ID." and "Only ID." columns). Regarding per-corner error, the results are also similar for every DeepArUco++ configuration, with marginally worse values for the configurations using the heatmap-based corner refinement model. However, this could stem from the fact that way more corners are being detected with this approach, possibly including some extreme cases not detected by the direct regression approach. The smallest per-corner error is achieved by the classic ArUco implementation; this could be the reason behind the almost perfect match between the number of detected markers and the number of markers with 4 corners remaining after filtering.

Figure 15: **Detail of some samples from the DeepTag ArUco dataset.** Note the presence of a certain overall "blockiness" and color noise around the edges of the markers and other elements in the scene. Best viewed in digital format.

Evaluation on DeepTag dataset. We have also tested our methods using a different dataset: in this case, we have considered the dataset used to develop DeepTag. A comparison of the detection capabilities of the different DeepArUco++ configurations against DeepTag on the DeepTag dataset (the portion of the `general` dataset containing ArUco markers) appears in Fig. 14b.

The results in Fig. 14b indicate that, under our test conditions, our method remains competitive on the DeepTag dataset, showing only a slight reduction in precision and recall when compared to the DeepTag method (when considering the `yolov8s` + heatmap configuration). Please note that we did not implement any specific measures to address particular characteristics of the DeepTag dataset, namely the noise and overall "blockiness" of the images (see Fig. 15), which may have reduced the detection capabilities of our models. A numerical comparison of the different methods on this new dataset appears in Table 3.

The data in Table 3 leads us to a similar conclusion to that obtained from Fig. 14b. When considering only marker detection, it seems like DeepTag is able to recover the most markers. However, the difference becomes much smaller when filtering corners far from their nearest ground-truth, especially when compared against the configurations using the `yolov8s`-based marker detector. On a curious note, it seems that both our baseline method and DeepTag achieve perfect precision when identifying markers (counting every marker for both "Corners + ID." and "Only ID." columns), regardless of the corner location accuracy. Regarding per-corner error, it seems that the configurations using the direct regression approach in the corner refinement step have the smallest error, with the rest of the methods presenting similar performance. Please note that for this dataset, precise marker locations were not provided, and we had to annotate the corner positions manually.

Cross-validation measurements. As an additional assessment of the performance of our methods, we have performed a 5-fold cross-validation, training and testing over portions of the Flying-ArUco v2 dataset. To this effect, we have split the Flying-ArUco v2 dataset into 5 different parts after augmenting its samples by using blur and color shift. Then, we trained our entire pipeline over 4 of such parts and tested the remaining. We repeated this process five

23

Table 3: **Quantitative comparison of DeepArUco++ detection capabilities against the state-of-the-art, on DeepTag dataset.** *Matched B.B.* stands for the percentage of markers detected, discarding markers with less than 4 detected corners at the output of the corner refinement step; *Corners filtered* stands for the percentage of detected markers that remain after discarding every corner for which the distance to the closest ground-truth corner is higher than 5 pixels, and keeping only markers with 4 remaining corners; *Corners + ID.* stands for the percentage of markers after filtering corners for which the predicted ID. is correct; *Only ID.* stands for the percentage of detected markers for which a correct ID. has been predicted, but less than 4 corners remain after filtering; *Corner error* stands for the mean Euclidean distance between a predicted corner and its nearest ground-truth corner (in pixels). For metrics noted with ↓, lower is better; for ↑, higher is better.

| Method | AUC ↑ | % of ground-truth markers | | | | Corner error ↓ |
		Matched B.B. ↑	Corners filtered ↑	Corners + ID. ↑	Only ID.	
yolov8n + direct	0.3972	59.37%	50.15%	50.14%	8.82%	2.18 ± 0.82
yolov8n + heatmap	0.4288	59.37%	51.74%	43.28%	7.23%	2.94 ± 0.82
yolov8s + direct	0.3967	65.23%	56.26%	55.22%	7.02%	**1.97 ± 0.84**
yolov8s + heatmap	**0.4785**	65.23%	59.79%	57.64%	3.60%	2.62 ± 0.81
Baseline (IbPRIA'23)	0.2760	64.00%	49.30%	49.30%	14.70%	2.70 ± 0.77
DeepTag	0.5583	**81.72%**	**63.36%**	**63.36%**	18.36%	2.49 ± 1.07

times for each configuration. The aggregate quantitative metrics appear in Table 4.

From the results in 4, we can see how for a given detector size (`yolov8n` or `yolov8s`) the choice of corner refinement model has the most impact in terms of both precise location and decoding performance, with the heatmap-based corner refinement model yielding the best results overall. Again, this would indicate that the heatmap-based approach manages to recover the most corners (even if the measured error *for recovered corners* is lower when using the direct-regression approach). On the other hand, the choice of marker detector size has not that much of an impact on this dataset.

5.5. Throughput comparison.

Finally, we can compare the different methods from a throughput standpoint. We have considered only two configurations, `yolov8n` detector with direct corner regression in the refinement step, and `yolov8s` detector with heatmap-based corner refinement. For both configurations, we have used the same marker decoding model. For DeepArUco++ configurations, we have timed the three stages: marker detection, corner regression (including the operations required to extract the final corner values when using a heatmap-based model), and marker

Table 4: **Quantitative comparison of multiple DeepArUco++ configurations, using 5-fold cross-validation.** *Matched B.B.* stands for the percentage of markers detected, discarding markers with less than 4 detected corners at the output of the corner refinement step; *Corners filtered* stands for the percentage of detected markers that remain after discarding every corner for which the distance to the closest ground-truth corner is higher than 5 pixels, and keeping only markers with 4 remaining corners; *Corners + ID.* stands for the percentage of markers after filtering corners for which the predicted ID. is correct; *Only ID.* stands for the percentage of detected markers for which a correct ID. has been predicted, but less than 4 corners remain after filtering; *Corner error* stands for the mean Euclidean distance between a predicted corner and its nearest ground-truth corner (in pixels). For metrics noted with ↓, lower is better; for ↑, higher is better.

Method	AUC ↑	% of ground-truth markers				Corner error ↓
		Matched B.B. ↑	Corners filtered ↑	Corners + ID. ↑	Only ID.	
yolov8n + direct	0.9182 ± 0.0484	99.44 ± 0.18%	96.07 ± 1.81%	72.81 ± 0.90%	1.75 ± 1.27%	1.47 ± 0.65
yolov8n + heatmap	0.9729 ± 0.0050	98.22 ± 0.38%	97.66 ± 0.43%	**74.71 ± 3.09%**	0.23 ± 0.08%	1.65 ± 0.60
yolov8s + direct	0.9170 ± 0.0521	**99.57 ± 0.08%**	96.23 ± 1.75%	72.74 ± 0.67%	1.76 ± 1.24%	**1.47 ± 0.63**
yolov8s + heatmap	**0.9742 ± 0.0042**	98.28 ± 0.31%	**97.73 ± 0.34%**	74.69 ± 3.09%	0.21 ± 0.08%	1.65 ± 0.59

decoding (including the time required to identify the marker ID). Additionally, we measure the time required for processing each frame (without including the time spent loading the image or writing the results). For DeepTag and classic ArUco, only the total time per frame has been measured. Results of this comparison appear in Table 5.

From data in Table 5, we can see how our methods achieve a significant throughput, with average per-frame times below 30 milliseconds. The high variance in time metrics reflects the challenges of accurately measuring short runtimes, which are affected by factors such as I/O activity and CPU load. The following conclusions can be drawn: first, there is little difference in inference times between using a detector based in `yolov8n` versus a detector based in `yolov8s`. The difference between `yolov8s` and `yolov8m` (as in the baseline DeepArUco) is more noticeable. On the corner refinement step, we can see a clear advantage in throughput when using the direct coordinate regression approach (as in the `yolov8n` + direct configuration, or the baseline DeepArUco) versus using the heatmap-based method. Finally, we can note how our methods outperform DeepTag from a throughput standpoint, with higher average FPS and way less variability.

5.6. Working examples and limitations.

In Fig. 16, we can see how both `yolov8s` + heatmap (a) and `yolov8n` + direct (b) manage to outperform classic ArUco (c) on the task of detecting ArUco markers in pitch-black conditions. This scene does not seem to pose any difficulties, with both configurations performing well enough, with `yolov8s` +

Table 5: **Time breakdown of the studied methods.** *Det. time* stands for the time spent per frame at the marker detection step (if applicable); *Ref. time* stands for the time spent at the corner refinement step; *Dec. time* stands for the time spent at the marker decoding step; *Total time* stands for the total time per frame (i.e. the time spent on every step, plus any extra time spent on operations not belonging to any step). Times are measured in milliseconds. Picture resolution is 800×600, with an average of 10.90 ± 1.33 markers per image. For metrics noted with ↓, lower is better; for ↑, higher is better.

Method	Det. time ↓	Ref. time ↓	Dec. time ↓	Total time ↓	FPS ↑
yolov8n + direct	**8.05** ± **7.63**	3.41 ± 5.39	13.37 ± 2.09	26.87 ± 12.98	37.78 ± 3.21
yolov8s + heatmap	8.08 ± 7.91	6.94 ± 4.90	**13.00** ± **1.42**	29.86 ± 12.55	33.88 ± 2.34
Baseline (IbPRIA'23)	9.77 ± 8.80	**3.28** ± **5.78**	13.44 ± 3.43	28.18 ± 14.64	36.46 ± 4.73
Classic ArUco	-	-	-	**8.57** ± **2.79**	**130.52** ± **44.62**
DeepTag	-	-	-	142.76 ± 114.16	28.66 ± 41.45

heatmap recovering an extra marker over yolov8n + direct. Please note that false detections may occur (e.g., the clock at the left of the scene). However, our method can discard most of these false detections by careful thresholding. On the other hand, DeepTag did not manage to recover any markers in this scene. Overall, from this image we can conclude that while difficult for the human eye, detecting and decoding markers in a darkened scene is feasible, provided the scene is uniformly dim.

While detecting markers on a uniformly darkened scene may seem difficult, detecting those in a scene with complex lighting patterns poses a more significant challenge, as they can mess with the edges of the markers, making the detection difficult. In Fig. 17, we can see how both variants of DeepArUco++ (a and b) manage to find and correctly decode more markers compared to both ArUco (c) and DeepTag (d). In this case, we can also see how the heatmap-based corner refinement model on yolov8s + heatmap (a) can outperform the direct regression method in yolov8n + direct (b), as exemplified by the marker with ID 242 in (a). The improvement in corner detection leads to a better crop, enabling the correct identification of the marker. Both classic ArUco (c) and DeepTag (d) fail to detect most markers in the scene.

Overall, it seems that detecting and decoding markers under complex patterns is indeed more difficult than under uniform lighting; this could stem from the effect of those patterns "flipping" some of the bits or at least reducing the certainty of their value (e.g., changing from black to a brighter tone in relation to the surrounding bits). If that is the case, it could be that our methods overly rely on local contrast to detect borders and extract the values of the bits in the marker. Taking this into account, much work could be done to improve detec-
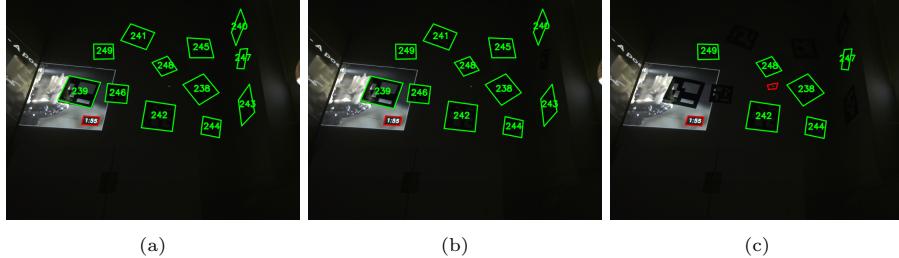
Figure 16: **Qualitative evaluation against SOTA methods, on Shadow-ArUco dataset on poor lighting conditions.** We have considered `yolov8s` + heatmap (a), `yolov8n` + direct (b), and classic ArUco (c). Red outlines indicate false positive detections. Red IDs indicate wrong identifications. (Best viewed in digital format)

tion and decoding by exploiting more high-level details, such as the expected shape of the marker, or even completely disregarding the bit-based approach, by interpreting complete bit "patterns" as individual pictographs.

In Fig. 18, we can see how the performance of the larger model is not always clearly superior to the smaller one. In this example, `yolov8s` + heatmap (a) has a similar performance to `yolov8n` + direct (b), only introducing a few additional errors. Scenarios like these are the most difficult for the method, by combining complex patterns with somewhat extreme marker poses.

Finally, in Fig. 19, we can see some additional, interesting failure cases that can be found when processing images with our methods. We could highlight the following behaviours: on the one hand, the configurations based on the larger detector (`yolov8s`) can be somewhat overzealous in scenarios with frequent contrasts between lights and shadows, discarding valid detections. This can be seen on Fig. 19a, when comparing the output from `yolov8n` + heatmaps (left) against the output from `yolov8s` + heatmaps (right). The explanation for this would probably stem from a slight overfitting to the training dataset, combined with an imperfect match between the conditions in our synthetic dataset and those in the test dataset. Additionally, our methods may hallucinate in pitch-black conditions, detecting markers where there are none (see Fig. 19b); in this case, the heatmap-based corner refinement model may lead to fewer false detections by not being able to find any corner in the image (right of Fig. 19b), while no detections will be discarded when using the direct regression approach (left of Fig. 19b). Finally, we can see how slight, nearly imperceptible variations in the cropping of a marker will lead to misidentifications: this can be seen in Fig. 19c when comparing the output for `yolov8n` + direct (left) against the output for `yolov8n` + heatmaps (right); additional work is needed towards getting a more predictable behaviour from our marker decoder.

5.7. Discussion of the Results.

From the results obtained from the experimentation in Sections 5.4 to 5.6, we can draw the following insights.
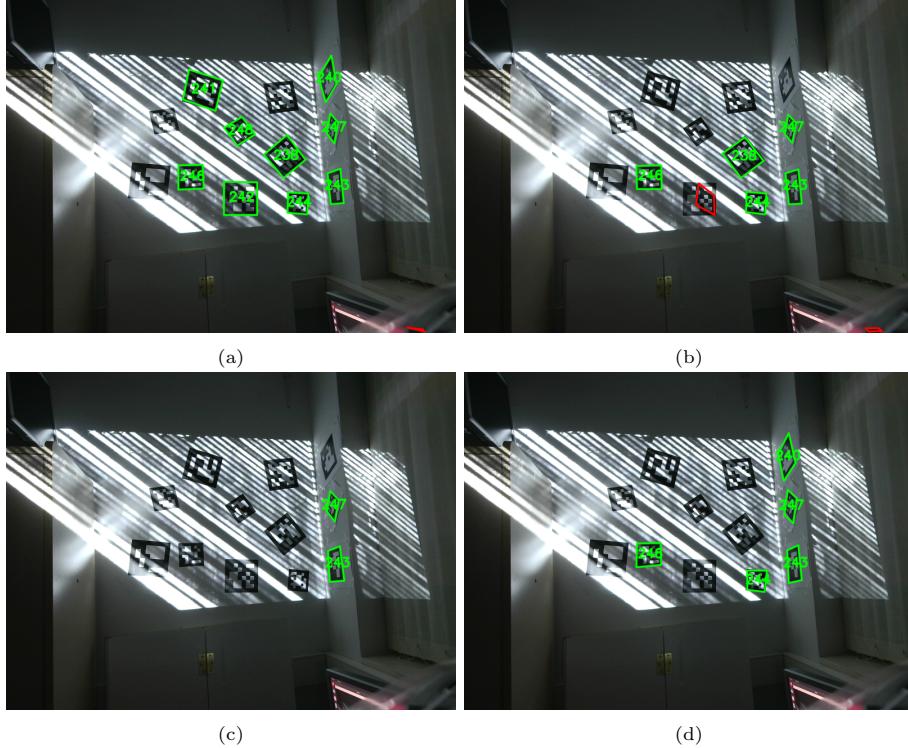
Figure 17: **Qualitative evaluation against SOTA methods, on Shadow-ArUco dataset with complex lighting patterns.** We have considered `yolov8s` + heatmap (a), `yolov8n` + direct (b), classic ArUco (c), and DeepTag (d). Red outlines indicate false positive detections. Red IDs indicate wrong identifications. (Best viewed in digital format)

Impact of detection model. The choice of marker detection model appears to have a relatively minor impact compared to the corner refinement approach. As shown in Tables 2 and 3, the most notable differences arise from using different corner refinement models rather than different detector models. This suggests that finding markers in an image is a much simpler task than extracting the precise location of their corners. During the corner refinement step, the direct regression approach is not that useful: while it will always return 4 predicted corners, they are often placed inaccurately, resulting in imprecise localization. While the error for the remaining corners after filtering is slightly lower with this approach, the heatmap-based approach consistently places more corners within an acceptable distance of their expected locations. Consequently, more markers survive the filtering step with the heatmap-based method.

Performance on DeepTag dataset. On the DeepTag dataset (see Table 3), some trends reverse. During the marker detection step, the configuration using `yolov8n` as the detector model and direct regression in the corner refinement step correctly decodes more markers than the one using `yolov8n` and
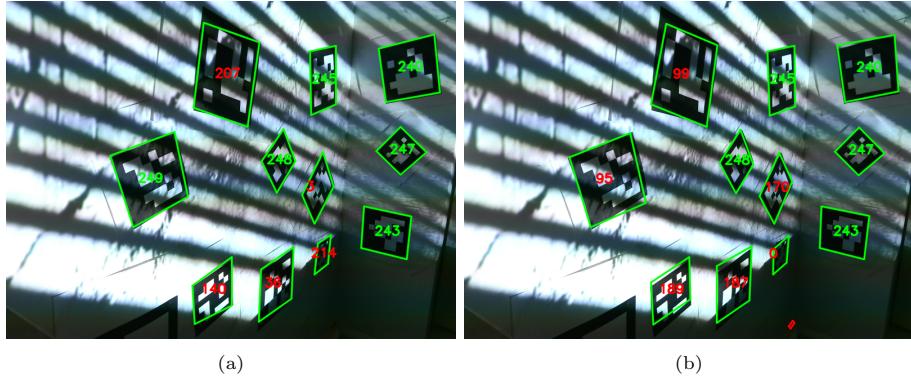
Figure 18: **yolov8s + heatmap vs. yolov8n + direct, on complex lighting settings.** `yolov8s` + heatmap on (a), `yolov8n` + direct on (b). Red outlines indicate false positive detections. Red IDs indicate wrong identifications. (Best viewed in digital format)

heatmap-based corner refinement. This discrepancy may be due to the higher corner error rates observed in the DeepTag dataset compared to the Shadow-ArUco dataset. However, these results might not fully reflect real-world performance due to certain irregularities in the DeepTag dataset, such as the use of the same marker in every image and the presence of "blockiness" in the images (see Fig. 15). Additionally, the dataset achieves perfect precision in marker decoding, which could indicate some degree of overfitting.

Throughput comparison. In terms of throughput (see Table 5), the classic ArUco implementation remains the best option, especially in environments with consistent and well-known lighting conditions. Our methods rank second, particularly when using the direct regression approach for the corner refinement step. The loss in throughput with the heatmap-based approach is mainly due to inefficiencies when extracting corner locations from heatmaps. DeepTag performs the worst, exhibiting lower average frames per second (FPS) and high variability in frame times, which makes it unsuitable for real-time applications.

Qualitative comparison. From a qualitative perspective, there is ample room for improvement in scenarios with frequent contrasts of light and shadows (see Fig. 17, 18 and 19a). Our method misses markers or fails to decode them correctly under these conditions. Accurately simulating the variability of real-world lighting in a synthetic dataset is challenging, and enhancing our understanding of the deployment environment could improve performance. Comparatively, detecting markers in poor but homogeneous light is a simpler task (see Fig. 16). Additionally, much work is needed towards reducing hallucinations in pitch-black environments (see Fig. 19b) and improving the consistency in the decoding step (see Fig. 19c).
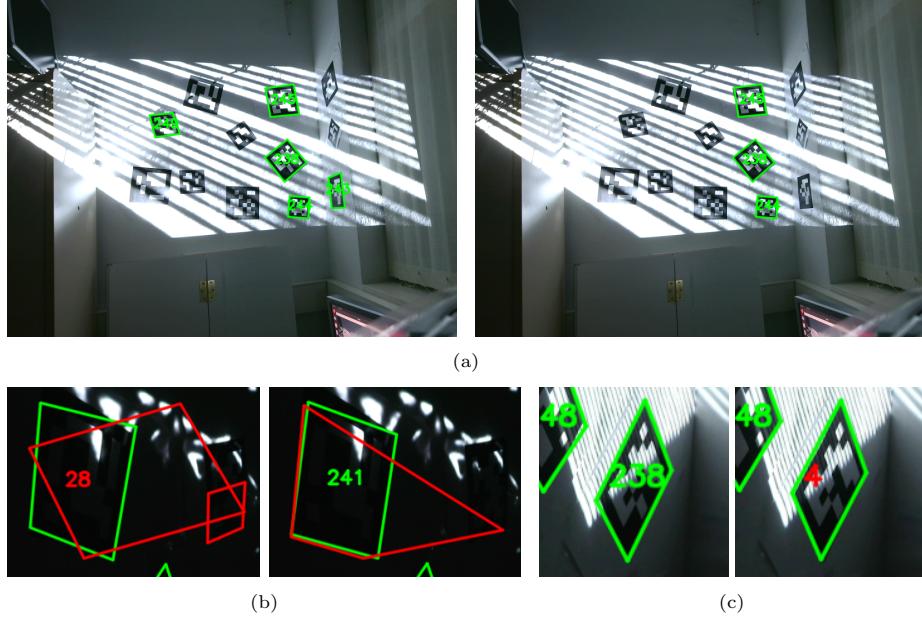
29

Figure 19: **Failure cases for multiple DeepArUco++ configurations.** The comparison between the multiple pairs of outputs in this figure highlights many deficiencies and failure cases that can be observed when using our methods. (a) A larger model can detect fewer markers than a smaller one: `yolov8n` + heatmaps (left) vs. `yolov8s` + heatmaps (right). (b) Models sometimes hallucinate in pitch-black conditions: `yolov8n` + direct (left) vs. `yolov8n` + heatmaps (right). (c) Small variations in predicted corners lead to wrong identifications: `yolov8n` + direct (left) vs. `yolov8n` + heatmaps (right). Red outlines indicate false positive detections and red IDs indicate incorrect identifications. Please refer to the main text for an in-depth explanation of the observed situations. (Best viewed in digital format)

6. Conclusions and Future Work

In this work, we have presented a methodology for detecting and decoding square fiducial markers in challenging lighting conditions. To do so, we have defined a pipeline composed of three steps: marker detection, corner refinement, and marker decoding, each based on a different neural network. The described approach is fully modular, allowing different combinations of models to get a configuration better suited for a given task.

To develop our methods, we have created two datasets: the first, fully synthetic (FlyingArUco-v2 dataset), has been used for training the different methods, and the second, obtained by recording physical markers under varying lighting settings (Shadow-ArUco dataset), has been used for testing and comparison against other state-of-the-art techniques. Additionally, we have used a third dataset (DeepTag) to provide a fair comparison outside our datasets.

Our method can outperform classic techniques such as classic ArUco in this task or even other state-of-the-art neural network-based techniques such as DeepTag. Additionally, our method can generalize to other domains outside

our target task (e.g., yielding competitive results on the DeepTag dataset, even when compared against the DeepTag method).

As a future work, the method we used to obtain the different models in this work could be extended to a broader range of situations, focusing on additional conditions outside of lighting settings; combining in-the-wild data with our synthetic dataset could also lead to a more robust method. Also, the impact of marker size, distance and angle with respect to the camera, and the presence of blur should be assessed. Finally, further optimization is needed on the implementation side to reduce the hardware requirements to deploy our methods, enabling competition against classic techniques in real-world applications.

Statements and Declarations

- **Competing interests:** The authors have no relevant financial or non-financial interests to disclose.

- **Ethics approval:** For this article, the authors did not undertake work that involved humans or animals.

- **Authors' contributions:** All authors contributed to the study conception and design. Material preparation, data collection, and analysis were performed by Rafael Berral-Soler. All authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

References

[1] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, M. J. Marín-Jiménez, Automatic generation and detection of highly reliable fiducial markers under occlusion, Pattern Recognition 47 (6) (2014) 2280–2292. `doi:10.1016/j.patcog.2014.01.005`.

[2] E. Olson, AprilTag: A robust and flexible visual fiducial system, in: 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 3400–3407. `doi:10.1109/ICRA.2011.5979561`.

[3] M. Fiala, Designing highly reliable fiducial markers, IEEE transactions on pattern analysis and machine intelligence 32 (2010) 1317–24. `doi:10.1109/ TPAMI.2009.146`.

[4] H. Sarmadi, R. Muñoz-Salinas, M. Álvaro Berbís, A. Luna, R. Medina-Carnicer, 3D Reconstruction and alignment by consumer RGB-D sensors and fiducial planar markers for patient positioning in radiation therapy, Computer Methods and Programs in Biomedicine 180 (2019) 105004. `doi: 10.1016/j.cmpb.2019.105004`.

[5] H. Sarmadi, R. Muñoz-Salinas, M. Álvaro Berbís, A. Luna, R. Medina-Carnicer, Joint scene and object tracking for cost-Effective augmented reality guided patient positioning in radiation therapy, Computer Methods and Programs in Biomedicine 209 (2021) 106296. `doi:10.1016/j.cmpb. 2021.106296`.

[6] M. F. Sani, G. Karimian, Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors, in: 2017 International Conference on Computer and Drone Applications (IConDA), 2017, pp. 102–107. `doi:10.1109/ICONDA.2017.8270408`.

[7] N. Strisciuglio, M. L. Vallina, N. Petkov, R. Muñoz-Salinas, Camera localization in outdoor garden environments using artificial landmarks, in: 2018 IEEE International Work Conference on Bioinspired Intelligence (IWOBI), 2018, pp. 1–6. `doi:10.1109/IWOBI.2018.8464139`.

[8] H. Sarmadi, R. Muñoz-Salinas, M. Álvaro Berbís, A. Luna, R. Medina-Carnicer, 3D Reconstruction and alignment by consumer RGB-D sensors and fiducial planar markers for patient positioning in radiation therapy, Computer Methods and Programs in Biomedicine 180 (2019) 105004. `doi: 10.1016/j.cmpb.2019.105004`.

[9] R. Muñoz-Salinas, R. Medina-Carnicer, UcoSLAM: Simultaneous localization and mapping by fusion of keypoints and squared planar markers, Pattern Recognition 101 (2020) 107193. `doi:10.1016/j.patcog.2019. 107193`.

[10] Z. Zhang, Y. Hu, G. Yu, J. Dai, DeepTag: A general framework for fiducial marker design and detection, IEEE Transactions on Pattern Analysis and Machine Intelligence 45 (03) (2023) 2931–2944. `doi:10.1109/TPAMI.2022. 3174603`.

[11] F. J. Romero-Ramírez, R. Muñoz-Salinas, R. M. Carnicer, Tracking fiducial markers with discriminative correlation filters, Image Vis. Comput. 107 (2021) 104094. `doi:10.1016/j.imavis.2020.104094`.

[12] D. Jurado-Rodriguez, R. Muñoz Salinas, S. Garrido-Jurado, R. Medina-Carnicer, Planar fiducial markers: a comparative study, Virtual Reality 27 (3) (2023) 1733–1749. `doi:10.1007/s10055-023-00772-5`.

[13] A. Krizhevsky, I. Sutskever, G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, in: Neural Information Processing Systems, Vol. 25, 2012. `doi:10.1145/3065386`.

[14] C. Shorten, T. M. Khoshgoftaar, A survey on Image Data Augmentation for Deep Learning, Journal of Big Data 6 (2019) 1–48. `doi:10.1186/s40537-019-0197-0`.

[15] R. Berral-Soler, R. Muñoz Salinas, R. Medina-Carnicer, M. J. Marín-Jiménez, DeepArUco: Marker detection and classification in challenging lighting conditions, in: Pattern Recognition and Image Analysis: 11th Iberian Conference, IbPRIA 2023, 2023, p. 199–210. `doi:10.1007/978-3-031-36616-1_16`.

[16] F. J. Romero-Ramirez, R. Muñoz-Salinas, R. Medina-Carnicer, Speeded up detection of squared fiducial markers, Image and Vision Computing 76 (2018) 38–47. `doi:10.1016/j.imavis.2018.05.004`.

[17] J. Wang, E. Olson, AprilTag 2: Efficient and robust fiducial detection, in: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016, pp. 4193–4198. `doi:10.1109/IROS.2016.7759617`.

[18] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, R. Medina-Carnicer, Generation of fiducial marker dictionaries using mixed integer linear programming, Pattern Recognition 51 (10 2015). `doi:10.1016/j.patcog.2015.09.023`.

[19] V. M. Mondéjar-Guerra, S. Garrido-Jurado, R. Muñoz-Salinas, M. J. Marín-Jiménez, R. M. Carnicer, Robust identification of fiducial markers in challenging conditions, Expert Syst. Appl. 93 (2018) 336–345. `doi:10.1016/j.eswa.2017.10.032`.

[20] N. Otsu, A threshold selection method from gray-level histograms, IEEE Transactions on Systems, Man, and Cybernetics 9 (1) (1979) 62–66. `doi:10.1109/TSMC.1979.4310076`.

[21] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. B. Zimmerman, K. Zuiderveld, Adaptive histogram equalization and its variations, Computer Vision, Graphics, and Image Processing 39 (3) (1987) 355–368. `doi:10.1016/S0734-189X(87)80186-X`.

[22] K. G. Lore, A. Akintayo, S. Sarkar, Llnet: A deep autoencoder approach to natural low-light image enhancement, Pattern Recognition 61 (2017) 650–662. `doi:10.1016/j.patcog.2016.06.008`.

[23] Y. Jiang, X. Gong, D. Liu, Y. Cheng, C. Fang, X. Shen, J. Yang, P. Zhou, Z. Wang, Enlightengan: Deep light enhancement without paired supervision, IEEE Transactions on Image Processing 30 (2021) 2340–2349. `doi:10.1109/TIP.2021.3051462`.

[24] D. Hu, D. DeTone, T. Malisiewicz, Deep ChArUco: Dark ChArUco marker pose estimation, in: 2019 IEEE/CVF Conference on Computer Vision and

Pattern Recognition (CVPR), 2019, pp. 8428–8436. `doi:10.1109/CVPR.2019.00863`.

[25] B. Li, J. Wu, X. Tan, B. Wang, Aruco marker detection under occlusion using convolutional neural network, in: 2020 5th International Conference on Automation, Control and Robotics Engineering (CACRE), 2020, pp. 706–711. `doi:10.1109/CACRE50138.2020.9230250`.

[26] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, J. Dai, Deformable detr: Deformable transformers for end-to-end object detection (2021). `doi:10.48550/arXiv.2010.04159`.

[27] Y. Cui, L. Yang, H. Yu, Learning dynamic query combinations for transformer-based object detection and segmentation, in: Proceedings of the 40th International Conference on Machine Learning, ICML'23, 2023.

[28] D. Meng, X. Chen, Z. Fan, G. Zeng, H. Li, Y. Yuan, L. Sun, J. Wang, Conditional detr for fast training convergence, in: 2021 IEEE/CVF International Conference on Computer Vision (ICCV), 2021, pp. 3631–3640. `doi:10.1109/ICCV48922.2021.00363`.

[29] X. Dai, Y. Chen, J. Yang, P. Zhang, L. Yuan, L. Zhang, Dynamic detr: End-to-end object detection with dynamic attention, in: 2021 IEEE/CVF International Conference on Computer Vision (ICCV), 2021, pp. 2968–2977. `doi:10.1109/ICCV48922.2021.00298`.

[30] G. Jocher, A. Chaurasia, J. Qiu, YOLO by Ultralytics (Jan. 2023).
URL https://github.com/ultralytics/ultralytics

[31] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, C. L. Zitnick, Microsoft COCO: Common Objects in Context, in: Computer Vision – ECCV 2014, 2014, pp. 740–755. `doi:10.1007/978-3-319-10602-1_48`.

[32] C. Poynton, Digital Video and HD: Algorithms and Interfaces, Morgan Kaufmann Series in Computer Graphics and Geometric Modeling, Elsevier Science, 2012.

[33] H.-S. Fang, S. Xie, Y.-W. Tai, C. Lu, RMPE: Regional multi-person pose estimation, in: 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2353–2362. `doi:10.1109/ICCV.2017.256`.

[34] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, M. Grundmann, Mediapipe: A framework for building perception pipelines (6 2019). `doi:10.48550/ARXIV.1906.08172`.

[35] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, 2015, pp. 234–241. `doi:10.1007/978-3-319-24574-4_28`.

[36] A. Newell, K. Yang, J. Deng, Stacked hourglass networks for human pose estimation, in: Computer Vision – ECCV 2016, 2016, pp. 483–499. `doi: 10.1007/978-3-319-46484-8_29`.

[37] K. Cosmas, A. Kenichi, Utilization of FPGA for Onboard Inference of Landmark Localization in CNN-Based Spacecraft Pose Estimation, Aerospace 7 (11) (2020). `doi:10.3390/aerospace7110159`.