Journal of Real-Time Image Processing (2021) 18:691–702
https://doi.org/10.1007/s11554-020-01010-w

ORIGINAL RESEARCH PAPER

PDCAT: a framework for fast, robust, and occlusion resilient fiducial marker tracking

Oualid Araar[1] · Imad Eddine Mokhtari[1] · Mohamed Bengherabi[1]

Received: 3 May 2020 / Accepted: 13 August 2020 / Published online: 25 August 2020

Abstract
Square binary patterns have become the de facto fiducial marker for most computer vision applications. Existing tracking solutions suffer a number of limitations, such as the low frame-rate and sensitivity to partial occlusions. This work aims at overcoming these limitations, by exploiting temporal information in video-sequences. We propose a parallel detection, compensation and tracking (PDCAT) framework, which can be integrated into any binary marker system. Our solution is capable of recovering markers even when they become mostly occluded. Furthermore, the low processing time of the tracking task makes PDCAT more than an order of magnitude faster than a track-by-detect solution. This is particularly important for embedded computer vision applications, wherein the detection run at a very low frame rate. In the experiments conducted on an embedded computer, the processing frame rate of the track-by-detect solution was merely 11 FPS. Our solution, on the other hand, was capable of processing more than 100 FPS.

Keywords Fiducial marker · Detection · Tracking · Embedded · Real-time · AprilTag

1 Introduction

Pose estimation is a fundamental problem in many computer vision applications, including robot navigation [1, 2], visual servoing [3, 4], augmented and virtual reality [5, 6]. The pose of a monocular camera can not be recovered unless a minimum set of correspondences between the 3D world and 2D image space is determined.

This can be achieved using natural features extracted online and matched against a 3D model of the scene. The 3D model can be obtained from CAD files or built offline using techniques such as structure from motion (SFM), or online using simultaneous localisation and mapping (SLAM) [7]. The advantage of using natural features is that no modification to the environment is required. The performance of such techniques, however, significantly degrades in poorly textured scenes. In this case, the usage of an active or passive fiducial marker becomes indispensable [8].

Active markers are designed using light emitting sources, commonly diodes, which can be easily segmented from the background. A problem with this solution is that the pose of an object cannot be obtained unless several markers are combined. This requires solving an assignment problem for identifying each marker. Another drawback is the limited number of patterns which can be achieved.

Passive markers can be designed based on colour information [9, 10], by building color patterns in such a way to make them distinguishable from the background. The usage of colour information permits a fast detection and identification. It is, however, sensitive to illumination conditions and is hence only advantageous in controlled environments. Shape patterns, on the other hand, are more robust to illumination changes but require more elaborate processing.

Among existing passive markers, square binary patterns have become the de facto solution for most computer vision applications. One of their advantages is that the 6 DOF pose of a calibrated camera can be retrieved from a single marker. Another attractive feature is the binary code they embed, which allows to deploy and distinguish a large number of patterns.

Binary marker design and detection have witnessed remarkable progress over the last decade. This can be noted not only in detection precision but also in the reduced computational complexity. This allows modern computers to run marker detection at relatively high frame rates. Nevertheless,

✉ Oualid Araar
oualid.araar@emp.mdn.dz; oualid.araar@gmail.com

1 Ecole Militaire Polytechnique, Bordj el Bahri, B.P 17, Algiers, Algeria

computation time is still considerable in systems with limited resources such as embedded computers and mobile devices. This is particularly worrying for use cases requiring pose estimation at high frame rates, such as visual servoing applications. Another major drawback of existing solutions is the failure of detecting partially occluded markers.

This work aims at overcoming these limitations by exploiting temporal information in video-sequences. Our contribution is a parallel detection and tracking framework, which allows recovering fiducial markers even when they become mostly occluded. Accurate localisation of fast moving markers is made possible thanks to a delay compensation strategy, which runs in parallel with the detection and tracking.

Another advantage of the proposed solution is the increased frame rate, brought about by the low computation time of the tracking task. In the experiments conducted, our solution is shown capable of running more than an order of magnitude faster than a track-by-detect approach.

The remainder of this paper discusses the rationale behind the proposed strategy and presents a set of experiments demonstrating its advantages. Section 2 presents existing works which are most relevant to our. Section 3 gives an overview of the proposed solution and its theoretical background. Section 4 discusses the architecture of the PDCAT framework and details its implementation. The experiments conducted and results obtained are discussed in Sect. 5.

2 Related work

Several marker designs have been proposed in the literature aiming to fulfil the conflicting requirements of precision and robustness. Depending on their geometry, existing markers can be classified into three major categories: circular [11–13], square [14–16] and topological patterns [17, 18].

Among existing solutions, those based on square forms have gained in popularity. These markers, originally designed for augmented reality applications, have been adopted in several other computer vision applications. These include mobile and aerial robotics, human machine interface, collaborative robotics, medical robotics, 3D reconstruction, to name but a few.

Square markers are composed of a black border to facilitate their detection and provide perspective support and minimal jitter [14]. Each marker embeds a unique code which defines its identifier.

Among early works, the ARToolKit [15] used custom patterns, such as Latin letters, to distinguish between different markers, Fig. 1. A detected marker is identified based on a template matching approach, by correlating it against a database of known patterns. This made the ARToolKit very sensitive to illumination conditions and perspective

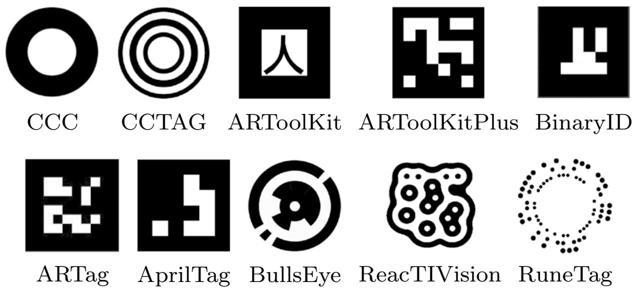

Fig. 1 Examples of fiducial markers proposed in the literature

distortion due to the difficulty of generating templates that are approximately orthogonal to each other.

To circumvent the limitations related to template matching, BinaryId [19] proposed patterns which implicitly encode the marker's id. ARtag [14, 20] followed on by introducing a built-in forward error correction. The ARtag's coding system consists of a six by six grid of bitonal cells in the interior of the marker. The digitally encoded payload of ARtag was later combined with ARToolKit resulting in the ARToolKitPlus [21]. Since no matching is required for binary markers, their detection is faster than template-markers.

The AprilTag [16, 22] is considered one of the most popular binary marker libraries that has been widely deployed in computer vision applications. It uses a graph-based image segmentation algorithm for a precise estimation of the marker's border. It also introduced a new coding system based on lexicodes to allow generating custom patterns with arbitrary sizes. Generated codes are parametrised by two quantities, the number of bits and the minimum Hamming distance between any two codewords. This solution explicitly guarantees the minimum Hamming distance for all four rotations of each tag and eliminates tags with low geometrical complexity.

ArUco [23] is another popular system which employs an adaptive detection threshold similar to AprilTag. Optimal codes, in terms of inter-marker-distance, are generated using a mixed integer linear programming solution. The work in [24] proposed a multi-scale approach to speed-up computation time of ArUco in high-resolution images. The adaptive threshold used in the original ArUco implementation is replaced by a global threshold. This further reduces processing time, but sacrifices the robustness in the face of illumination conditions.

In most computer vision applications, fiducial markers appear in video sequences. Nevertheless, to our knowledge, their tracking has only been realized using track-by-detect solutions. Frame to frame tracking is generally faster than tracking by detection due to the limited search zone. The accuracy of frame to frame tracking, however, degrades with time due to error accumulation.

The proposed solution combines detection and tracking in a single framework to maintain the advantages of the two approaches and reduce their drawbacks. Our work can be seen as an improvement which can be integrated into any binary marker system. Before discussing the implementation of the proposed strategy, we present an overview of the tracking solution we adopted, and justify its use in the PDCAT solution.

3 Feature detection and tracking solution

The objective of the tracking is to localize an initial set of features or regions in an incoming sequence of images. State of the art solutions are based on deep learning techniques [25–30]. These works target the tracking of objects of general appearances, i.e. objects with no a priori knowledge.

Fiducial markers have a particular appearance which favours the usage of simpler and faster tracking techniques. As will be demonstrated hereafter, binary patterns are rich in terms of corners that are very advantageous for tracking using the famous KLT tracker (Lucas and Kanade [31], Shi and Tomasi [32]). This tracker is known for its rapidity, which we exploit to increase the limited frame rate of existing marker detectors.

In what follows, an overview of the selected tracker is presented to show the source of its efficiency. After that, we demonstrate the richness of binary markers in terms of features which ensure the best tracking performances. This justifies the choice of the KLT tracker for implementing the PDCAT solution.

3.1 Feature tracking

The KLT tracker exploits local intensity information to direct the search for correspondences. This results in a much faster convergence compared to brute-force techniques [31]. For this algorithm to work properly, each two consecutive images have to verify a certain constancy in terms of pixel brightness and spatial coherence.

Tracking a point, p, from the previous image, I_p, to the current image, I_c, corresponds to the determination of the displacement vector $d[d_x\ d_y]$ such that $\big(I_p(p) = I_c(p+d)\big)$. In the KLT algorithm, the displacement vector d is chosen so as to minimize the error defined by the following double integral over the tracking window W [32]:

$$\epsilon(d) = \sum\sum_{(x,y)\in W} \big(I_p(x,y) - I_c(x+d_x, y+d_y)\big)^2. \tag{1}$$

Under the small displacement vector assumption, the intensity function can be approximated by its first order Taylor series expansion. The residue of Eq. (1) can hence be written as

$$\epsilon(d) = \sum\sum_{(x,y)\in W} \left(I_p(x,y) - I_c(x,y) - \begin{bmatrix} \frac{\partial I_c}{\partial x} & \frac{\partial I_c}{\partial y} \end{bmatrix} d\right)^2. \tag{2}$$

Since ϵ is a quadratic function of d, its minimisation can be achieved in closed form, by setting its first derivative w.r.t d to zero, which yields

$$\sum\sum_{(x,y)\in W} \left(I_p(x,y) - I_c(x,y) - \begin{bmatrix} \frac{\partial I_c}{\partial x} & \frac{\partial I_c}{\partial y} \end{bmatrix} d\right) \begin{bmatrix} \frac{\partial I_c}{\partial x} & \frac{\partial I_c}{\partial y} \end{bmatrix} = 0. \tag{3}$$

By assuming a constant displacement d inside the tracking window, one can write

$$Gd = e. \tag{4}$$

For every pair of adjacent frames, matrix G can be computed from one frame, by estimating gradients (I_x, I_y) and computing their second-order moments,

$$G = \sum\sum_{(x,y)\in W} \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}. \tag{5}$$

The main issue with the original KLT tracker is its limited convergence region. To overcome this problem, Bouguet [33] proposed a pyramidal implementation. This is achieved by iteratively running the original KLT solution on the different pyramid levels. The result at a each level is passed to the next one and used as an initial guess, which is further refined using the same window size.

3.2 Corner detection

Given the a priori knowledge about the tag's form, one can detect its inside corners by wrapping the detected edges based on the estimated pose. This approach resembles that used by classical chessboard based calibration techniques, where the user selects the four external corners, and the remaining corners are calculated from the prior knowledge. Such a solution, however, can not ensure precise detection of the corners [34].

It then becomes necessary to consider a corner detection solution, which selects pixels that maximize a certain measure of cornerness. One of the classical measures of cornerness is the Moravec algorithm [35], which tests how similar a patch centred on a pixel to its nearby centred patches, using a sum of squared differences (SSD)

$$\mathrm{SSD}(d_x, d_y) = \sum\sum_{(x,y)\in W} w(x,y)(I(x+d_x, y+d_y) - I(x,y)), \tag{6}$$

with $w(x, y)$ a weighting function, commonly a Gaussian. The main issue with the Moravec operator is that its response is anisotropic, given that only a discrete set of shifts is assessed. The operator also responds too readily to edges, since the SSD minimum is the only criterion applied [36].

The work of Harris and Stephens [36] improved Moravec's algorithm and resulted in the well-known Harris corner detector. This is achieved using an analytic expansion of the SSD in order to cover all possible shifts, however, small. As by definition a corner leads to a large variation of the SSD in all directions, the cornerness in Harris' algorithm is directly related to the eigenvalues of matrix G (Eq. 5).

To avoid the complexity of calculating eigenvalues, the algorithm suggests the following measure of cornerness, which is calculated from the determinant and trace of matrix G,

$$M_C = \lambda_1\lambda_2 - \Lambda_H(\lambda_1 + \lambda_2)^2 = \det(G) - \Lambda_H \mathrm{trace}^2(G). \qquad (7)$$

In accordance with Harris' expansion idea, Shi and Tomasi [32] demonstrated that selecting corners based on their minimum eigenvalue leads to features which are most suitable for tracking using KLT tracker. In fact, for the estimated displacement to be accurate, matrix G has to be well conditioned and above the image noise level. This is equivalent to its both eigenvalues being large. It follows that features which ensure a good tracking are those satisfying a large minimum eigenvalue.

Figure 2 illustrates the corners detected using the good feature to track (GFTT) solution proposed by Shi and Tomasi [32] for different marker families. This confirms our hypothesis of the richness of binary markers in terms of features suitable for tracking using the KLT tracker.

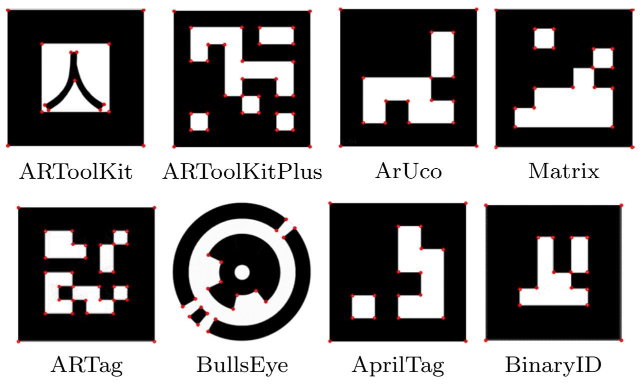

Fig. 2 Corners detected using the good feature to track solution for different marker families. This justifies the choice of the KLT tracker for implementing PDCAT

4 PDCAT framework

In what follows, the implementation of the PDCAT solution is discussed in more detail. We start with a basic version which consists of just the detection and tracking tasks. The purpose of presenting this preliminary version is to facilitate understanding the final PDCAT solution and to demonstrate the necessity of the compensation task mentioned earlier.

4.1 Parallel detection and tracking (PDAT)

The first version, PDAT, is composed of two main threads: a detection thread and a tracking thread which run in parallel.

4.1.1 Detection thread

This thread points to the last captured image and detects existing markers using the original implementation of AprilTag version 2.0. If at least one marker is detected the "Det_End" flag is triggered to communicate the end of a successful detection to the tracking thread. The corners inside the detected tags are calculated using the GFTT solution. Before communicating them to the tracking task, these corners are concatenated with the four external corners returned by the detection algorithm.

4.1.2 Tracking thread

This thread applies the pyramidal KLT tracker to the corners returned by the detection task. The tracking is continued on the upcoming frames until a new detection becomes available ("Det_End" flag set to true). At this point, the tracking is reinitialised to take into account the new positions of the tracked corners. This way, we avoid accumulation of the tracking error which may grow unbounded, especially in the case of a fast motion of the marker.

Since the tracking runs much faster than the detection, the former will be able to treat all incoming images, while the latter may miss an important number of frames.

The PDAT solution has proved efficient for slowly moving markers. In the case of rapid motion, however, we noticed a significant degradation in the performance of the algorithm. This is due to the important difference between the marker's position at detection start (frame n in Fig. 3) and its position in the frame at which the tracking is reinitialized (frame $n + k$).

In fact, if the marker undergoes a considerable motion between frame n and frame $n + k$, the position of the features returned to the tracking thread will considerably differ from their position in the current image. This will affect the quality of the tracking in two aspects. The first is a bias which

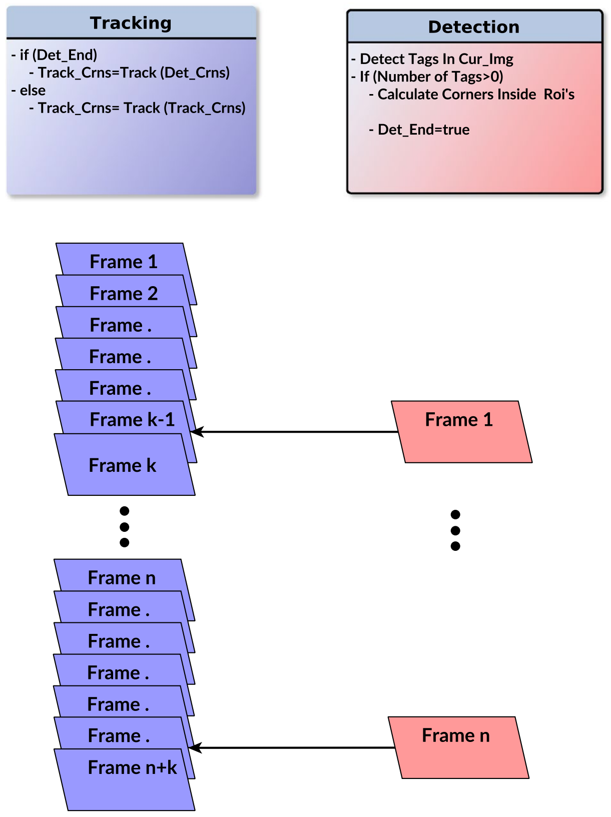

Fig. 3 Illustration of the first version PDAT

will be added to the tracking error until a new detection is returned. The second, and more dangerous problem, is that the important difference between corner positions at time n and $n+k$ will violate the assumption of small motion, necessary for the KLT solution to ensure an accurate tracking.

Figure 4 illustrates this problem, where the blue square represents the marker position in the detection frame, and the red square is the position estimated by the tracking thread. As can be observed, the detection lag leads to important errors in the returned corner positions, which will further degrade the tracking quality for the reasons explained previously.

4.2 Parallel detection, compensation and tracking

The idea of PDCAT is to save the sequence of frames that follows the detection frame, and to track back the features returned by the detection task over this sequence. This way, we make sure that the motion of the marker between the detection frame and the current frame, whatever its amount, is taken into account.

A possible option for implementing this solution is to realize the compensation by the tracking thread. This additional task, however, results in an extra load on this thread, which will cause it to miss upcoming frames while being busy with the compensation. To avoid this problem, a dedicated thread is added. Its role is to compensate for the detection delay, hence the name compensation thread.

The second version PDCAT is composed of four threads, in addition to an optional display thread. The detection thread is identical to that of the PDAT version. The remaining threads are discussed in what follows.

Figure 5 illustrates the work-flow of PDCAT and the variables communicated between the different threads. The frame sequences placed under the threads indicate the end of the processing of each frame. This gives a qualitative view about the processing time of each task and the synchronisation in the PDCAT solution.

4.2.1 Image acquisition and storage thread

This thread captures new images, converts them to grey scale, stamps them with an incremental ID and then stores them in an image stack. To avoid memory growth, the image stack size has to be kept to a minimum, by deleting old frames. A frame is not deleted until we make sure that it is not being processed by the detection nor the tracking threads, i.e

$$(\text{ID} < \text{Track\_ID}) \text{ and } (\text{ID} < \text{Det\_ID}). \tag{8}$$

We also make sure that this frame does not belong to a sequence being processed by the compensation thread, i.e

$$\text{not}(\text{Comp\_Idle}) \text{ and } (\text{ID} < \text{Comp\_Ref\_ID}). \tag{9}$$

The “Comp_Ref_ID” refers to the last frame with successful detection, while the “Comp_Idle” flag indicates whether a compensation is currently running. This condition ensures

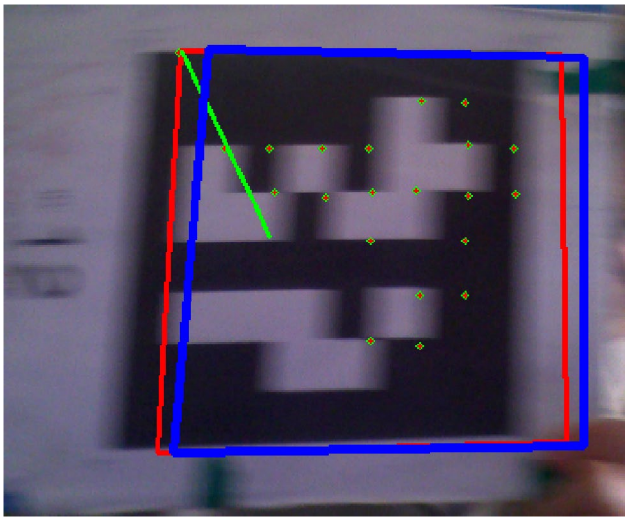

Fig. 4 Illustration of the problem of PDAT version. Note the difference between the actual positions of the corners and the positions returned by the detection thread

avoiding memory growth in the case of a long sequence with no successful detection.

4.2.2 Compensation thread

This thread plays the role of an intermediary between the detection and tracking tasks. Whenever a new detection becomes available, “Det_End” flag set to true, the compensation thread copies the corners returned by the detection task (“Det_Crns”). It then starts tracking them over the sequence of images that followed the frame in which the last successful detection was performed, “Det_ID”.

Since the compensation thread runs faster than the tracking thread, it will be able to overtake it. The tracking is stopped as soon as the compensation thread reaches the ID being treated by the tracking thread. At this point, the flags “Comp_End” and “Comp_Idle” are triggered to communicate the end of compensation to the tracking thread and the image storage thread, respectively.

4.2.3 Tracking thread

This thread is implemented in the same way as the first version (PDAT). The only difference is that it communicates with the compensation thread instead of the detection one. It tracks the corners, “Track_Crns”, in the last captured image, “Cur_Img”, and reinitializes its corner vector whenever a new compensation is completed, “Comp_End=true”.

4.3 Robust corner matching

In its form presented earlier, the tracking task returns the positions of the corners in the current image. This is, however, insufficient for estimating the pose of the marker since the positions of these corners in the real world are not available. To solve this issue, we combine all the corners to estimate the affine transformation between the current frame, and the last detection frame, for which the pose of the marker is available.

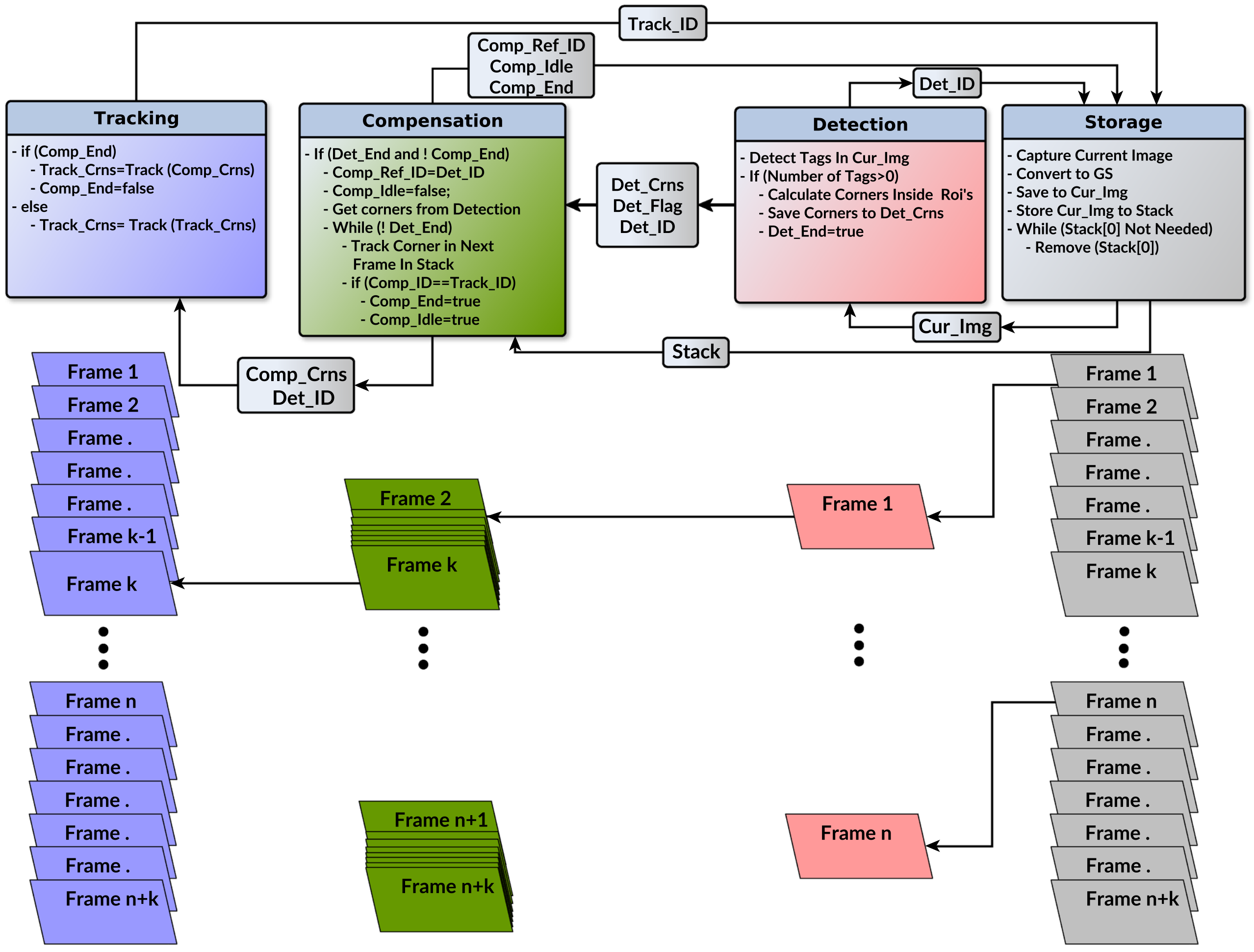

Fig. 5 Flowchart of the implementation of the second version PDCAT

The estimated transformation will permit recovering the positions of the four external corners of the marker in the current image, and hence estimating the pose of the marker. Given that the features of interest belong to the same plane, it can be demonstrated that the corners in the detection frame $c_d[u_d\ v_d]$ and their coordinates in the current frame $c_c[u_c\ v_c]$ are related by a planar Homography matrix [37]:

$$\begin{bmatrix} \lambda u_c \\ \lambda v_c \\ \lambda \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} u_d \\ v_d \\ 1 \end{bmatrix} \tag{10}$$

Since the number of tracked features may change over time, either due to occlusions or tracking loss, the vector c_d has to be updated accordingly.

Another problem which may occur, especially in the case of abrupt motion of the maker, is the presence of outliers among the tracked features. Including them in the calculation of the Homography matrix may lead to a completely erroneous estimation of the marker pose.

The M-estimator and RANdom SAmple Consensus (RANSAC) are the most common solutions to ensure a robust estimation in the presence of outliers. In this work, the RANSAC is adopted to detect outliers. The remaining inliers are used for estimating the Homography matrix.

5 Experiments and results

This section describes the experiments conducted to evaluate the proposed PDCAT solution, mainly in terms of tracking accuracy and processing time. The results presented hereafter correspond to the C++ implementation on a general purpose laptop with the characteristics summarized in Table 1.

To demonstrate the advantage of the increased frame rate brought about by our solution, we also evaluate its implementation on an embedded single board computer. The board we used is the Odroid XU4, illustrated in Fig. 6. The main characteristics of this board are summarized in Table 1.

In what concerns the software, both computers run an ubuntu operating system. The OpenCv library is exploited for implementing the tracking solution. The detection routine is conducted using the original implementation of the AprilTag library version 2.0. For the parallel implementation, the POSIX library is used. The five threads are created using a function pointer to each of the five tasks discussed previously. The threads are initialised in the same order in which they appear in Fig. 5, followed by the display thread. The acquisition, detection and display threads start with the first incoming frame, while the compensation and tracking threads are triggered by the activation of the flags "Det_End" and "Comp_End" respectively.

Table 1 Specifications of the computers used for evaluating the proposed PDCAT solution

	Processor	Memory (GB)	Graphics	O.S
Laptop	Intel Core i5-4340M CPU @ 2.90GHz × 4	4	GeForce GTX 860M	64-bit
Embedded computer	Cortex-A15 2.0 GHz Cortex-A7 1.4 GHz	2	Mali-T628	32-bit

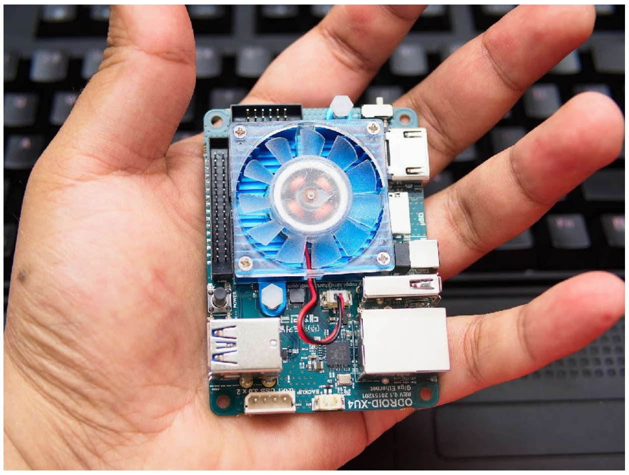

Fig. 6 Embedded computer, Odroid XU-4, used for validating the proposed PDCAT solution

Figure 7 illustrates the capacity of the proposed solution in handling important occlusions of the marker. Indeed, since the internal corners of the marker are exploited in the tracking, its position can be recovered as long as at least four corners are visible. This represents the minimum number of correspondences necessary for estimating the Homography matrix.

As depicted in Fig. 7, the detection solution fails in recovering the marker position as soon as one of its external corners is occluded. Our solution, on the other hand, was capable of tracking the marker even when it becomes mostly occluded.

In what follows, we compare quantitatively the results obtained using PDCAT to those of the track-by-detect solution. To make the comparison possible, one has to ensure repeatability by running the two implementations on the same video-sequences recorded beforehand. These sequences feature different scene textures, lighting conditions and motion patterns, and total a number of 20,234 frames.

The tracking error reported in what follows is calculated as the average Euclidean distance between the marker position estimated by PDCAT and the position returned by the detection solution. Since the detection misses a large number of frames, the comparison is conducted with detection results obtained offline.

After a thorough analysis of the results obtained using PDCAT, it was concluded that the main parameters which affect its performances are the size of the tracking window and the number of pyramid levels (when using the pyramidal implementation). Extensive tests have been conducted to evaluate the influence of these parameters and to select the optimal values, in terms of processing time and tracking accuracy.

It is to note that the results presented hereafter are obtained by testing the PDCAT solution on the total number of images (i.e 20234 frames) for each pair of these parameters. Regarding the number of pyramid levels, we considered six values, from 0, which represents the original KLT implementation, to five levels. The tracking window size was varied from 3 to 21 with a step of 2.

Figure 8 plots the tracking error, calculated as the average mean square error between the four corners of the marker returned by the detection solution and those estimated using PDCAT. Both the results achieved on the laptop and the embedded computer demonstrate the advantage of the pyramidal implementation, especially for a small size of the tracking window. Nevertheless, for a sufficiently large tracking window, the error of the classical KLT tracker becomes comparable to the pyramidal implementation.

Figure 9 depicts the average processing time for the tracking thread of the PDCAT solution and the processing time of the track-by-detect solution. As can be noticed from the figure, the processing time for the track-by-detect solution largely exceeds that of PDCAT, for all the combinations of window size and number of pyramids we tested.

Figure 10 plots in more detail the advantages of the lower processing time of our solution, in terms of frame rate gain. As shown in this figure, PDCAT is more than an order of magnitude faster than the track-by-detect solution. The resulting frame rate, which varies from around 300 FPS to more than 500 FPS for the results obtained on the laptop, largely exceeds the actual frame rate for most ordinary cameras. Nevertheless, the ameliorations brought to embedded applications is clearly of interest.

Indeed, the processing frame rate of the track-by-detect solution running on the embedded computer is merely 11 FPS. Our PDCAT solution, on the other hand, can achieve more than 100 FPS when using the non-pyramidal implementation. The lowest achieved frame rate using the pyramidal implementation still largely exceeds that of the track-by-detect solution.

From these results, one can clearly notice the important increase in the processing time of PDCAT, when using the pyramidal implementation. Since the classical KLT tracker yields a good precision for a sufficiently large tracking window, we opted for this implementation to benefit from its lower processing time.

With a tracking window of 11×11 pixels^2, one can achieve more than 500 FPS on the laptop and more than 100 FPS on the embedded computer. In what follows, we present more details about the results obtained using the selected pair of parameters, i.e 11×11 pixels^2 tracking window and 0 pyramid levels.

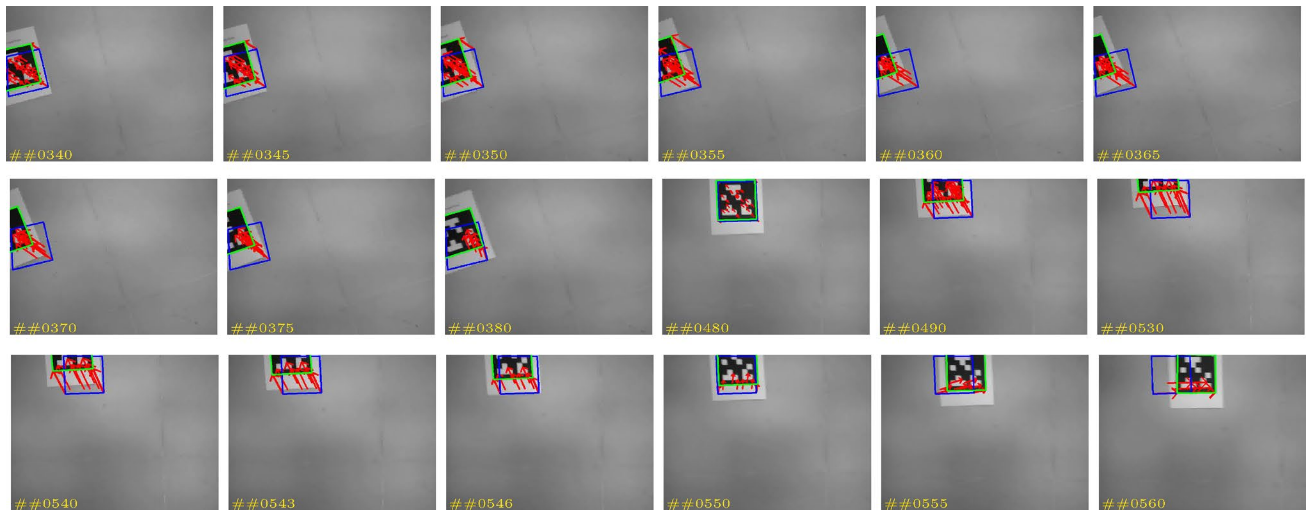

Fig. 7 An application scenario illustrating the advantage of the proposed PDCAT solution in handling partial occlusions of the marker. The blue square indicates the last position in which the detection solution was capable of recovering the marker. The green square represents the marker position estimated using our solution

Figure 11 plots the tracking results for one of the video-sequences used in our evaluation. From this figure, one can notice that the marker trajectory returned by PDCAT coincides exactly with the ground truth trajectory, obtained from the detection.

The error between the two trajectories are plotted in Fig. 12. The maximum tracking error for this scenario is less than four pixels for the laptop implementation and slightly exceeds four pixels for the implementation on the embedded computer.

From Fig.13, which plots the normalised histogram for the tracking error, one can notice that these errors are concentrated between zero and one pixel, meaning that our solution yields sub-pixel accuracy.

Since the rapidity of any image processing solution depends on the size of the images, we run another set of experiments to evaluate the processing time of PDCAT and the track-by-detect solution in terms of the size of the input image.

To ensure the repeatability of the conducted test, we resized a selection of the video-sequences used in the previous experiments to different image resolutions, and tested PDCAT as well as the track-by-detect solution following the same previous procedure.

Figures 14 and 15 plot respectively the average processing time and frame rate for the different image resolutions we tested. As can be observed from these figures, the gap between the two solutions increases with the size of input images, thing which further motivate our solution for higher image resolutions.

The processing time of the track-by-detect solution exceeds 40 ms on the laptop and 200 ms on the embedded computer for a resolution of 1200x900. The PDCAT average processing time, on the other hand, remains less than

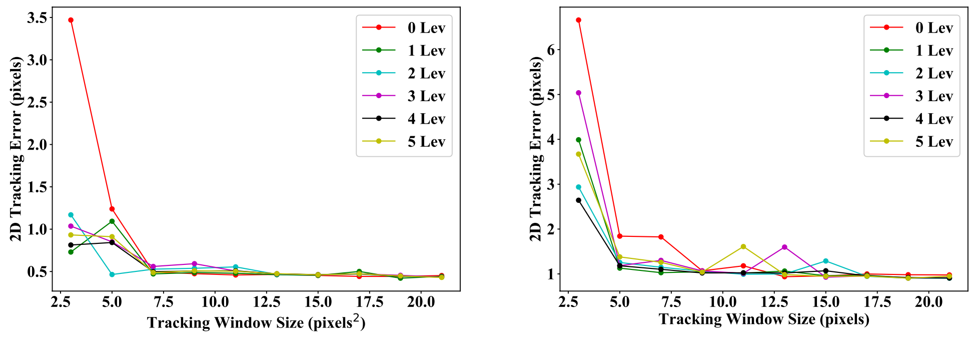

Fig. 8 Tracking error measured as the average Euclidean distance between the four corners estimated using PDCAT and their position returned by the AprilTag library

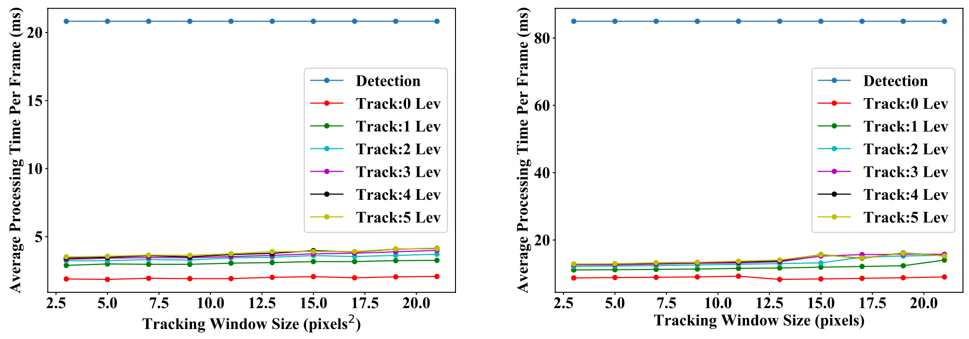

Fig. 9 Average Processing time for the PDCAT framework and the track-by-detect solution

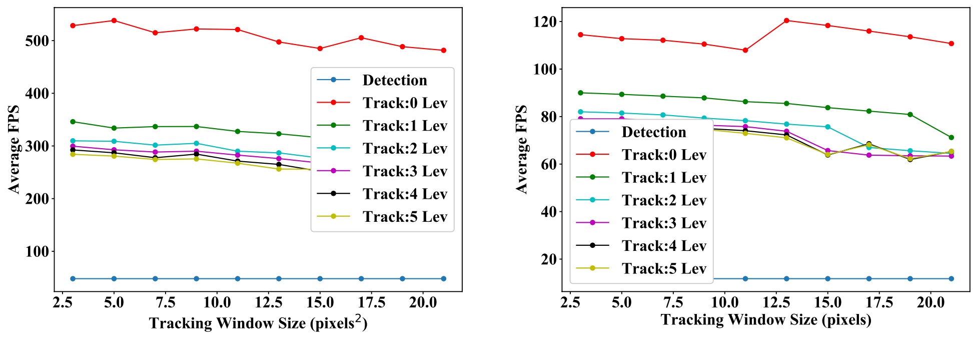

Fig. 10 Average processing rate achieved using PDCAT compared to the average frame rate of the detection solution: left (Experiments on Laptop); right (experiments on embedded computer)

5 ms for the laptop and less than 30 ms for the embedded computer.

6 Conclusion and future work

This work proposed a solution for adapting fiducial markers to real-time computer vision applications. The proposed approach exploits the richness of binary markers in terms of corners, which makes their tracking using the famous KLT tracker very robust. This tracker is exploited in a parallel detection compensation framework, to allow the tracking of even fast moving objects.

In the experiments conducted on a general purpose laptop and an embedded computer, the proposed PDCAT solution permitted more than an order of magnitude increase in the processing frame rate. This allowed bringing the few FPS frame rate of the track-by-detect solution to more than 100 FPS, when running on the embedded computer.

In addition to the reduced computation time, the PDCAT framework was capable of recovering almost completely occluded markers, thanks to the exploitation of the internal corners of the marker.

Future work will target more optimisation of the current PDCAT implementation, and its evaluation on other embedded computers. The investigation of a solution for adapting the parameters of the tracking task, depending on the pose of the marker might also enhance the accuracy of the tracking. In fact, the current parameters of the algorithm are selected so as to optimise the average tracking performance over different distances and orientations of the marker. These parameters, namely the size of the tracking window, might be adjusted depending on the size and pose of the marker.

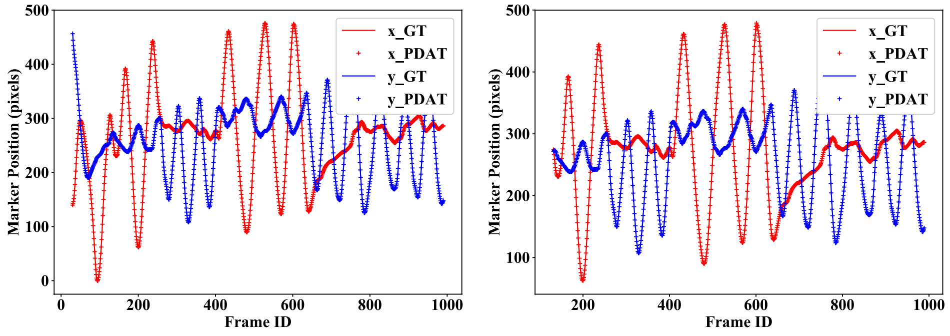

Fig. 11 Marker 2D position estimated using PDCAT solution (red), and the track-by-detect solution (blue)

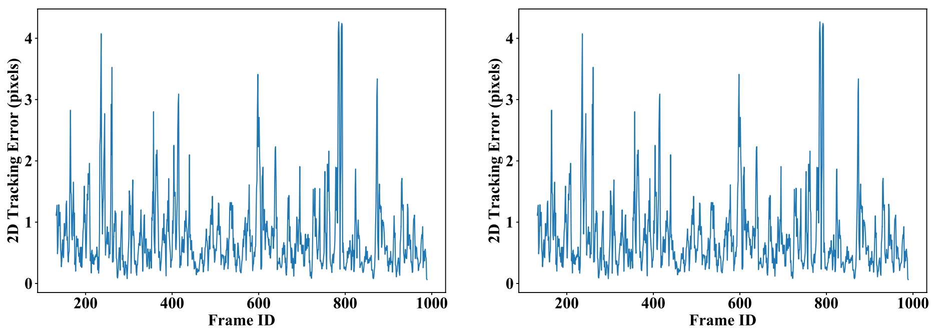

Fig. 12 Tracking error for one of the video sequences we used in the evaluation of PDCAT

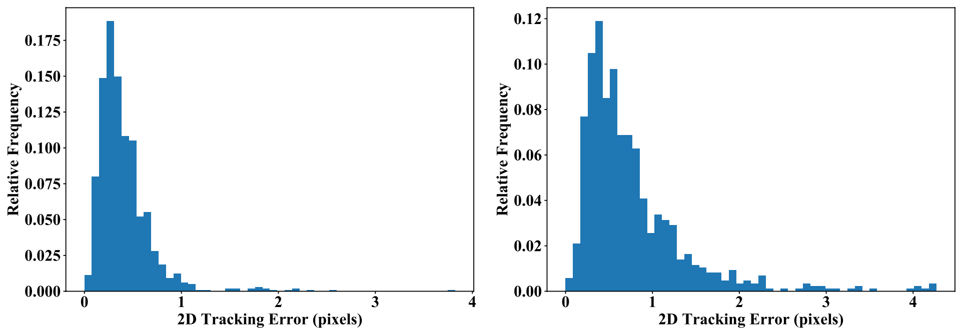

Fig. 13 Normalised histogram of the tracking error, left (experimentations on laptop), right (experimentations on embedded computer)

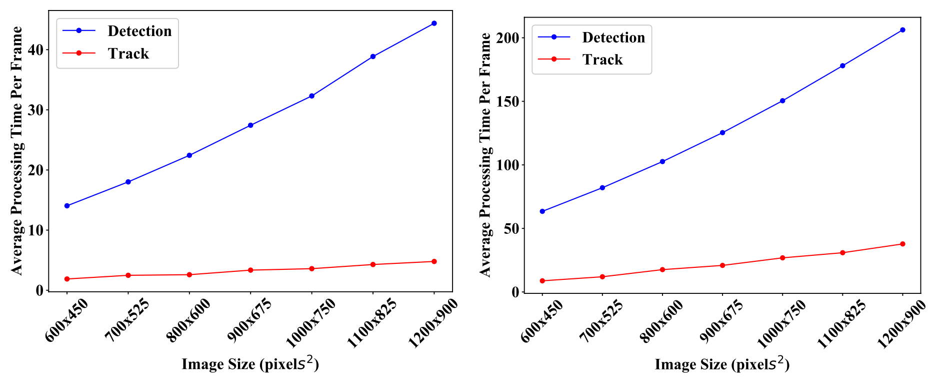

Fig. 14 Average processing time for the PDCAT solution and the track-by-detect solution as a function of image resolution

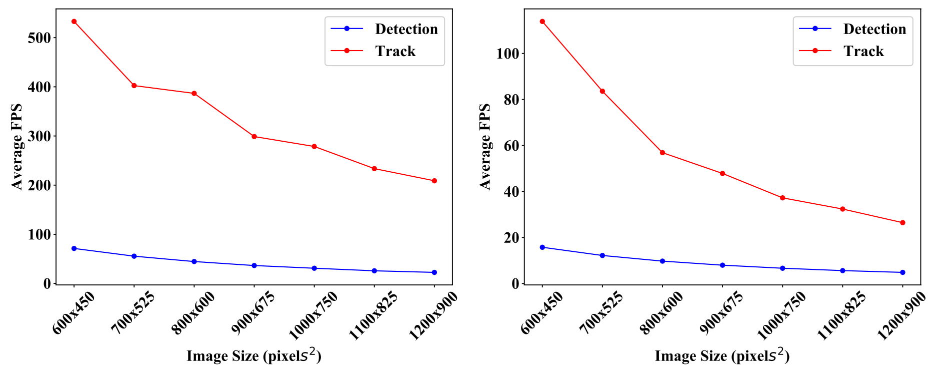

Fig. 15 Average frame rate for the PDCAT solution and the original detection solution as a function of image resolution: left (experimentations on laptop); right (experimentations on embedded computer)

The usage of the PDCAT framework for the tracking of other objects is another point to be looked at. This will necessitate investigating the adequate tracking solution which ensures the required rapidity and robustness.

References

1. La Delfa, G.C., Monteleone, S., Catania, V., De Paz, J.F., Bajo, J.: Performance analysis of visualmarkers for indoor navigation systems. Front. Inf. Technol. Electron. Eng. **17**(8), 730–740 (2016)
2. Spławski, M., Staszak, R., Jarecki, F., Chudziński, J., Kaczmarek, P., Drapikowski, P., Belter, D.: “Motion planning of the cooperative robot with visual markers. In: *Conference on Automation*, Springer, pp. 206–215 (2020)
3. Sivčev, S., Rossi, M., Coleman, J., Dooly, G., Omerdić, E., Toal, D.: Fully automatic visual servoing control for work-class marine intervention rovs. Control Eng. Pract. **74**, 153–167 (2018)
4. Araar, O., Aouf, N., Vitanov, I.: Vision based autonomous landing of multirotor uav on moving platform. J. Intell. Robot. Syst. **85**(2), 369–384 (2017)
5. Barandiaran, I., Paloc, C., Graña, M.: Real-time optical markerless tracking for augmented reality applications. J. Real Time Image Process. **5**, 129–138 (2010)
6. Park, J., Seo, B.-K., Park, J.-I.: Binocular mobile augmented reality based on stereo camera tracking. J. Real Time Image Process. **13**, 571–580 (2017)
7. Chandaria, J., Thomas, G.A., Stricker, D.: The matris project: real-time markerless camera tracking for augmented reality and broadcast applications. J. Real Time Image Process. **2**, 69–79 (2007)
8. Santos, P.C., Stork, A., Buaes, A., Pereira, C.E., Jorge, J.: A real-time low-cost marker-based multiple camera tracking solution for virtual reality applications. J. Real Time Image Process. **5**(2), 121–128 (2010)
9. Lampert, C.H., Peters, J.: Real-time detection of colored objects in multiple camera streams with off-the-shelf hardware components. J. Real Time Image Process. **7**, 31–41 (2012)
10. Bagherinia, H., Manduchi, R.: Robust real-time detection of multi-color markers on a cell phone. Real Time Image Process. **8**(2), 207–223 (2013)
11. Gatrell, L.B., Hoff, W.A., Sklair, C.W.: Robust image features: concentric contrasting circles and their image extraction. In: Stoney, W.E. (ed.) Cooperative Intelligent Robotics in Space II, vol. 1612, pp. 235–244. International Society for Optics and Photonics, Bellingham (1992)
12. Calvet, L., Gurdjos, P., Griwodz, C., Gasparini, S.: Detection and accurate localization of circular fiducials under highly challenging conditions. In: 2016 IEEE Conference on cmputer vision and pattern recognition (CVPR), pp. 562–570, (2016)
13. Bergamasco, F., Albarelli, A., Rodolà, E., Torsello, A.: Rune-tag: a high accuracy fiducial marker with strong occlusion resilience. CVPR **2011**, 113–120 (2011)
14. Fiala, M.: Designing highly reliable fiducial markers. IEEE Trans. Pattern Anal. Mach. Intell. **32**(7), 1317–1324 (2010)
15. Kato, H., Billinghurst, M.: Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In: Proceedings on 2nd IEEE and ACM International Workshop on Augmented Reality. (IWAR ’99), pp. 85–94, (1999)
16. Olson, E.: AprilTag: a robust and flexible visual fiducial system. In: 2011 IEEE International conference on robotics and automation, IEEE, pp. 3400–3407, (2011)

17. Klokmose, C.N., Kristensen, J.B., Bagge, R., Halskov, K.: Bullseye: high-precision fiducial tracking for table-based tangible interaction. In: Proceedings of the Ninth ACM international conference on interactive tabletops and surfaces, pp. 269 – 278, (2014)
18. Kaltenbrunner, M., Bencina, R.: Reactivision: a computer-vision framework for table-based tangible interaction. In: Proceedings of the 1st international conference on Tangible and embedded interaction, pp. 69–74, (2007)
19. Flohr, D., Fischer, J.: A Lightweight ID-based extension for marker tracking systems. In: Froehlich, B., Blach, R., van Liere, R. (eds.) Eurographics Symposium on Virtual Environments, Short Papers and Posters. The Eurographics Association, Switzerland (2007)
20. Fiala, M.: ARTag, a fiducial marker system using digital techniques. In: IEEE computer society conference on computer vision and pattern recognition, pp. 590–596, (2005)
21. Wagner, D., Schmalstieg, D.: ARToolKitPlus for pose tracking on mobile devices ARToolKit. In: Computer vision winter workshop, (2007)
22. Wang, J., Olson, E.: AprilTag 2 : efficient and robust fiducial detection. In: International conference on intelligent robots and systems, pp. 4193–4198, (2016)
23. Garrido-jurado, S.: Automatic generation and detection of highly reliable fi ducial markers under occlusion. Pattern Recogn. **47**(6), 2280–2292 (2014)
24. Romero-ramirez, F.J., Mu, R.: Speeded up detection of squared fiducial markers. Image Vis. Comput. **76**, 38–47 (2018). https://doi.org/10.1016/j.imavis.2018.05.004
25. Ciaparrone, G., Sánchez, F.L., Tabik, S., Troiano, L., Tagliaferri, R., Herrera, F.: Deep learning in video multi-object tracking: a survey. Neurocomputing **381**, 61–88 (2020)
26. Bertinetto, L., Valmadre, J., Henriques, J.F., Vedaldi, A., Torr, P.H.: Fully-convolutional siamese networks for object tracking. In: European conference on computer vision, Springer, pp. 850–865, (2016)
27. Huang, K., Shi, Y., Zhao, F., Zhang, Z., Tu, S.: Multiple instance deep learning for weakly-supervised visual object tracking. Signal Process. Image Commun. **84**, 115807 (2020)
28. Chu, J., Tu, X., Leng, L., Miao, J.: Double-channel object tracking with position deviation suppression. IEEE Access **8**, 856–866 (2019)
29. Yuan, Y., Chu, J., Leng, L., Miao, J., Kim, B.-G.: A scale-adaptive object-tracking algorithm with occlusion detection. EURASIP J. Image Video Process. **2020**(1), 1–15 (2020)
30. Chu, J., Guo, Z., Leng, L.: Object detection based on multi-layer convolution feature fusion and online hard example mining. IEEE Access **6**, 19959–19967 (2018)
31. Lucas, B., Kanade, T.: An iterative image registration technique with an application to stereo vision. In: 7th international joint conference on artificial intelligence, vol. 130, pp. 674–679, (1981)
32. Shi, J., Tomasi, C.: Good features to track. In: IEEE computer society conference on computer vision and pattern recognition CVPR, pp. 593–600, (1994)
33. Bouguet, J.: Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. Intel corporation (2001)
34. Chu, J., GuoLu, A., Wang, L.: Chessboard corner detection under image physical coordinate. Opt. Laser Technol. **48**, 599–605 (2013)
35. Moravec, H.: Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover. Carnegie-Mellon University, Pittsburgh (1980)
36. Harris, C., Stephens, M.: A combined corner and edge detector. In: Procedings of the Alvey vision conference 1988, pp. 23.1–23.6, (1988)
37. Márquez-Neila, P., López-Alberca, J., Buenaposada, J.M., Baumela, L.: Speeding-up homography estimation in mobile devices. J. Real Time Image Process. **11**, 141–154 (2016)

Oualid Araar was born in Algeria in 1987. He received the M.Sc. degree in Electrical Engineering in 2011 from the Ecole Militaire Polytechnique. He received the Ph.D. degree in Automation and Control in 2015, from Cranfield University, UK. He is currently a lecturer at the Ecole Militaire Polytechnique. His research interests are mainly within the areas of computer vision, visual servoing, and perception for robotics.

Imad Eddine Mokhtari was born in Algeria in 1994. He received the Bachelor degree in 2016 and the M.Sc. degree in Electrical Engineering in 2019, both from the Ecole Militaire Polytechnique. His research is focused on real-time computer vision and image processing.

Mohamed Bengherabi was born in Algeria in 1994. He received the Bachelor degree in 2016 and the M.Sc. degree in Electrical Engineering in 2019, both from the Ecole Militaire Polytechnique. His research interests include visual navigation, perception and control.