2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

CopperTag: A Real-Time Occlusion-Resilient Fiducial Marker

Xu Bian[1,†], Wenzhao Chen[2,†,*], Xiaoyu Tian[2] and Donglai Ran[2]

***Abstract*— Fiducial markers, like AprilTag and ArUco, are extensively utilized in robotics applications within industrial environments, encompassing navigation, docking, and object grasping tasks. However, in contrast to controlled laboratory conditions, markers installed in factory grounds or equipment surfaces, often face challenges like damage or contamination. These issues can lead to compromised marker integrity, resulting in reduced detection reliability. To address this challenge, we propose a novel fiducial marker called CopperTag, which incorporates circular and square elements to create a robust occlusion-resistant pattern. The CopperTag detection process relies on three fundamental steps: firstly, extracting all lines from the image; secondly, identifying corners; and lastly, searching for quadrilateral candidate regions using ellipses and nearby corners. The Reed-Solomon (RS) algorithm is utilized for both encoding and decoding the information content. This algorithm possesses the ability to recover corrupted messages in situations where CopperTag data is incomplete. The experimental results illustrate that CopperTag exhibits superior robustness and accuracy in detection when compared to other state-of-the-art fiducial markers, even in scenarios with heavy occlusion. Moreover, CopperTag maintains an average processing time of 10ms per frame on a standard laptop, effectively meeting the real-time demands of robotics applications.**

I. Introduction

Fiducial markers play an important role in robotics applications owing to their capacity to provide enhanced stability and robust features in complex environments when compared to natural features [1]. This significance becomes especially apparent in industrial settings, where tasks such as autonomous mobile robot (AMR) goods transportation [2], [3], docking to conveyor ports [4], and object grasping by mobile manipulator (MOMA) [5] demand millimeter-level precision in positioning. Existing fiducial markers encompass Quick Response (QR) codes, Data Matrix (DM) codes, AprilTag, ArUco, among others. QR codes and DM codes primarily serve as tools for encoding and transmitting substantial amounts of information. In contrast, markers like AprilTag and ArUco find their primary application in positioning, owing to their simple design. However, employing these existing markers in industrial robotics applications presents several challenges:

- **Pattern-Dependency:** Many existing fiducial markers heavily depend on specific finder patterns, such as convex quadrilaterals, homocentric squares, or L-shapes. Any form of damage or occlusion affecting these patterns can lead to detection failures, which is a prevalent issue in industrial environments.
- **Occlusion-Complexity:** QR codes, DM codes and STag can handle partial occlusion to some extent. However, real-world occlusion scenarios are more intricate, affecting both inner and outer areas with varying degrees of corner damage. These markers are unable to address the full spectrum of diverse occlusion scenarios effectively.

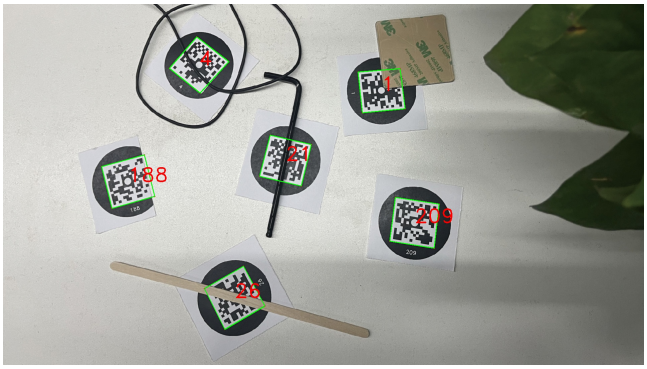


Fig. 1. CopperTag detection example: The CopperTags were occluded by various objects, yet they were all successfully detected

To solve these issues, we introduce CopperTag, a novel fiducial marker that combines circular and square to create a robust occlusion-resistant pattern, as shown in Figure 1. Our contributions include addressing prevalent industrial occlusion scenarios with CopperTag's ability to manage the absence of up to two corners and tolerate up to 30% incomplete information area. We also provide open access to the CopperTag system code [1], fostering advancements in fiducial marker research.

II. Related Work

In industrial scenarios, fiducial markers serve two primary purposes: information storage and positioning. Figure 2 showcases the commonly used markers alongside the proposed CopperTag. Information storage markers such as QR codes [6] and DM codes [7] employ RS algorithm to encode extensive data, offering high information density and resilience against occlusion. However, incomplete finder patterns in these markers can disrupt the detection process.

The use of square markers for positioning provides a straightforward detection method and ensures precise corner position. ARToolKit [8], introduced in 1999, features various inner patterns but suffers from high false positives and

† These authors contributed equally.
* Corresponding author email: robotmanciu@gmail.com.
[1] School of Mechanical Engineering, Xi'an Jiaotong University, Xi'an, 710049, China.
[2] Research and Development Department, Shenzhen Youibot Robotics Co. Ltd., Shenzhen, 518100, China.

[1]https://github.com/ManciuCen/CopperTag

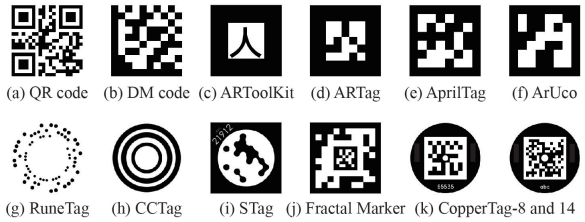


Fig. 2. The fiducial markers commonly mentioned in the literature, including the proposed CopperTag

inner-marker confusion. In contrast, ARTag [9], based on ARToolKit, employs digital coding theory and gradient-based line detection, enhancing detection reliability even with partial occlusions.

AprilTag [10] builds upon ARTag, introducing a novel coding system and gradient pattern segmentation. Subsequent iterations like AprilTag2 [11] and AprilTag3 [12] refine its capabilities, with faster detection and flexible marker designs. ArUco [13], based on ARTag and ARToolKit, offers reliable open-source marker detection with customizable dictionaries. ArUco3 [14], an improved version, speeds up detection using down-sampling technique but assumes a convex quadrangle outer border, leading to detection failures with occlusion or damage.

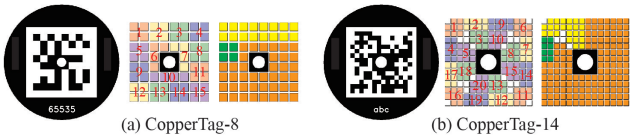
STag [15] is a recently developed fiducial marker system that combines squares and circles to create a more stable marker. Its primary contribution lies in achieving precise results in homography calculation, thereby enhancing the accuracy of camera pose estimation. To address occlusion challenges, certain fiducial markers have been proposed. RuneTag [16], a circular marker, offers the flexibility of having either one or three concentric circular sections, providing robustness against partial occlusions. However, RuneTag's detection speed is relatively slow, presenting difficulties in real-time robotics applications. On the other hand, CCtag [17], a round monochromatic marker with concentric black and white rings, achieves reliable detection even in conditions of motion blur and occlusion. Nevertheless, CCtag lacks any ID encoding feature, thereby limiting its use in large-scale manufacturing factories.

An alternative solution to overcome occlusion problems involves partitioning the marker into outer and inner components. This design enables the inner component to remain functional when the outer component is obscured, and vice versa. HArCO [18] proposes a general approach to constructing hierarchical structures within square markers, resulting in the creation of sub-markers. Similar techniques include Fractal Marker [19], a novel marker type formed by aggregating squared markers nested within each other recursively, with multiple markers sharing the same reference point. However, these markers do not fully resolve the issue of occlusion splitting down the middle, leading to a contamination of both inner and outer markers.

III. CopperTag system design

A. CopperTag definition

CopperTag employs a composite design featuring a circular and a square shape, effectively forming a robust external pattern serving as fiducial markers. An illustration of CopperTag can be found in Figure 3, including CopperTag-8 and CopperTag-14. The numeral accompanying CopperTag indicates the dimensions of the inner data area. For example, the label "8" signifies an 8x8 RS data area, with an additional surrounding blank pixel, resulting in an overall inner square of 10x10 pixels. Positioned within this inner square is a strategically placed 2x2 pixel circle, which serves to enhance marker center location accuracy, particularly in "same-side" occlusion scenarios.



(a) CopperTag-8 (b) CopperTag-14

Fig. 3. The definitions of Copper-8 and CopperTag-14 are illustrated in Figure (a) and Figure (b). Figure (a) presents a configuration featuring an 8x8 inner information area, while Figure (b) showcases a 14x14 inner information area. In the middle image of both (a) and (b), the numbers above the regions indicate the decoding order. Moving to the right image in both (a) and (b), the yellow-colored regions are designated for data symbols, the orange-colored regions for parity symbols, and the green-colored regions for customization

B. Detection method

In the initial phase of the detection process, we begin by downsampling the input image to reduce computational time. Following this, an adaptive threshold method is employed for binarizing the input image. Subsequently, all edge segments are extracted using the algorithm proposed by Suzuki [20]. These edge segment groups serve as the input for the arc adjacency matrix-based method (AAMED) [21], which is used to identify potential elliptical regions.

The edge segments extracted by Suzuki algorithm [20] contain only cluster information, they cannot differentiate between lines or corners within the clusters. The subsequent crucial step involves extracting all lines from the aforementioned edge segment groups and generating corners at the intersection of two adjacent lines. We proposed a new algorithm, detailed in Algorithm 1, to address this process. During this procedure, all edge segments are traversed, and the Douglas-Peucker (DP) algorithm is applied to approximate the input edge segment into polygon with fewer vertices. The approximation results in the creation of salient points along the edge, denoted as contour. The accuracy of continuous line contour consisting of more than two salient points and exceeding a length of 14 pixels (value τ) can be enhanced through the introduction of line refinement method. As illustrated in Figure 4(a), a representative example demonstrates the transformation of a square pattern into a trapezoidal pattern, often attributed to limitations in low-resolution cameras or long-distance imaging. The single-shot polygon approximation method may yield erroneous

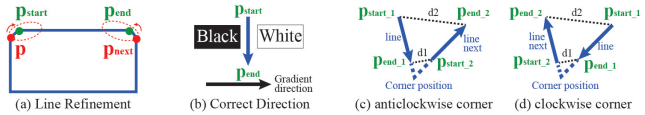
Algorithm 1: Extract lines and form corners

Input: edge segment groups $edges$
Output: corner groups $corners$

foreach $edge \in edges$ **do**
  $contour$ = Approximate-Polygon($edge$)
  if *$contour$ is long enough* **then**
    foreach p *in* $contour$ **do**
      if *Distance(p, p_{next}) > τ* **then**
        p_s, p_e = Refine-Line(p, p_{next})
        $gradient$ = Get-Gradient($edge$)
        Correct-Direction($gradient$, p_s, p_e)
        $lines \leftarrow$ Line(p_s, p_e)
      end
    end
  end
end
foreach $l \in lines$ **do**
  if l *is long enough* **then**
    if *Angle(l, l_{next}) > θ* **then**
      pos = Intersection(l, l_{next})
      d_1 = Distance($l.p_e$, $l_{next}.p_s$)
      d_2 = Distance($l.p_s$, $l_{next}.p_e$)
      if $d_1 < d_2$ *and* $d_1 < \psi$ **then**
        if *Cross-Product(l, l_{next}) < 0* **then**
          $corners \leftarrow$ Corner(pos, l, l_{next})
        end
      else if $d_2 < d_1$ *and* $d_2 < \psi$ **then**
        if *Cross-Product(l_{next}, l) < 0* **then**
          $corners \leftarrow$ Corner(pos, l_{next}, l)
        end
    end
  end
end
return $corners$;

contour points, as indicated by the red points in Figure 4(a), which fail to accurately represent the actual lines in the image. The DP algorithm is once again applied during the line refinement process, but this time within a limited range of 7 pixels. The computation starts from the current processing point and extends towards the midpoint of the $p - p_{next}$ line. The selection of the penultimate DP contour point within the range of p is denoted as p_{start}, while the penultimate DP contour point within the range of p_{next} is denoted as p_{end}.

The subsequent step involves adjusting the positions of points p_{start} and p_{end} to ensure that the line $p_{start} - p_{end}$ is oriented in such a way that its right side corresponds to the black region, while its left side corresponds to the white region, as illustrated in Figure 4(b). This adjustment is performed by evaluating the sign of the cross product between the vector representing the line $p_{start} - p_{end}$ and the gradient direction of points along this line. If the sign is negative, then no changes are made to the current orientation; otherwise, points p_{start} and p_{end} are swapped.



(a) Line Refinement (b) Correct Direction (c) anticlockwise corner (d) clockwise corner

Fig. 4. The crucial steps in Algorithm 1. Figure(a) illustrates the line refinement process, the red arrow denotes the search direction, and the dashed ellipse outlines the search range. Figure(b) illustrates that a valid line definition in Algorithm 1 should have its right side in black and its left side in white. Figure (c) and (d) display the same corer but in anti-clockwise and clockwise order, respectively. In algorithm 1, all corners are consistently saved in an anti-clockwise order.

We utilize the extracted lines to form corners. Specifically, we calculate the included angle between a given line and its subsequent counterpart. Angles exceeding 22.5 degrees (value θ) are identified as potential corners. Additionally, we compute distances denoted as d_1 and d_2. d_1 corresponds to the distance between the p_{end} of $line_1$ and the p_{start} of $line_2$, while d_2 pertains to the distance between the p_{start} of $line_1$ and the p_{end} of $line_2$. The determination of which line is designated as $line_{in}$ and which is $line_{out}$ depends on the proximity of points p_{end} and p_{start} to the intersection point. If d_1 is smaller than d_2 and is smaller than 7 pixel (value ψ), $line_1$ assumes the role of $line_{in}$, with $line_2$ taking that of $line_{out}$, and vice versa. For a corner to be used in the post-process, it must exhibit a negative sign in the cross product between $line_{in}$ and $line_{out}$. This attribute aligns with the definition of an anti-clockwise corner, as shown in Figure 4(c).

Following the identification of candidate ellipses and corners, the search for CopperTag candidates is conducted by traversing through the ellipses and corners. We propose an algorithm, outlined in Algorithm 2, to describe this process, primarily aiming to estimate convex quadrilaterals. The estimation method presented in Algorithm 2 is designed to handle various occlusion scenarios. We have summarized four common occlusion scenarios encountered in industrial settings, as illustrated in Figure 5. These occlusion scenarios include the triangle, same-side, symmetry, and diagonal cases, each characterized by the number of input corners and their positional relationships.

The missing corner in the triangle scenario is estimated by calculating the intersection of the $line_{in}$ from the first input corner and the $line_{out}$ from the last input corner. In the same-side scenario, two corners are missing on one edge. We estimate the marker's diagonal line using the ellipse center as a reference point, and then restore the two missing corners by calculating the intersection of the diagonal line and the $line_{in}$ or $line_{out}$ from the input corner. Each same-side case temporarily stores current input corners and checks if they can form a quadrilateral by merging with another facing same-side corner group after traversing all corners close to the current ellipse, as shown in line 19 of Algorithm 2. Two cases of the same-side that can be merged correspond to the symmetry scenario. Conversely, the diagonal case can only be processed during isolated corner estimation, as represented in line 16 of Algorithm 2. In this case, we recover

Algorithm 2: CopperTag candidate search algorithm

Input: corner groups $corners$, ellipse groups $ellipses$
Output: CopperTag candidate groups $markers$

foreach $corner \in corners$ **do**
  if *corner.size() >= 3* **then**
    $markers \leftarrow$ Estimate-Quads($corner$)
  end
end
foreach $ellipse \in ellipses$ **do**
  foreach $corner \in corners$ **do**
    if *Distance(corner, ellipse)* $> \alpha$ **then**
      continue
    end
    $validCorner \leftarrow$ Extracted($corner$)
    $markers$, $isolatedCorner \leftarrow$ Estimate-Quads($validCorner$)
  end
  Reorder-Filter-Corners($isolatedCorner$);
  if $isolatedCorner.size()$ *>= 2* **then**
    $markers \leftarrow$ Estimate-Quads($isolatedCorner$)
  end
  Replace-Fake-Corners($markers$, $isolatedCorner$)
  Merge-Sameside-Quads($markers$)
  Remove-Duplicated-Quads($markers$)
end
return $markers$

the two missing corners by calculating the intersection points of the two lines of the two input corners respectively.

In the initial phase of the CopperTag candidate search algorithm, we evaluate all corners with more than three corners to determine if they can directly form a quadrilateral or if they should follow the triangle case. Following this, each ellipse is utilized as an anchor for searching nearby corners. Corners that are in close proximity to the ellipse are accepted, while small and slanted corners are filtered out. For the remaining corners, we once again use the "Estimate-Quads" method to generate quadrilaterals based on their positional relationships and number. Valid corners that cannot form a quadrilateral are grouped as isolated corner groups. After traversing all corner groups, these isolated corner groups may contain numerous corners. To handle this, we create a convex hull to extract the outermost corners and arrange them in an anti-clockwise order. Finally, if there are more than one isolated corners present, we perform one last check for quadrilateral formation using the "Estimate-Quads" method.

In cases where individual isolated corners cannot be grouped with others to form a quadrilateral, we access if they can replace some estimated corners in existing quadrilaterals. The satisfactory condition for replacement is when the distance between the isolated corner and current corner is less than 5 pixels, while their corresponding lines are nearly parallel. Ultimately, we eliminate duplicated quadrilaterals within one ellipse based on their estimated corner count and perimeter, retaining quadrilaterals with more actual corners and a larger perimeter.

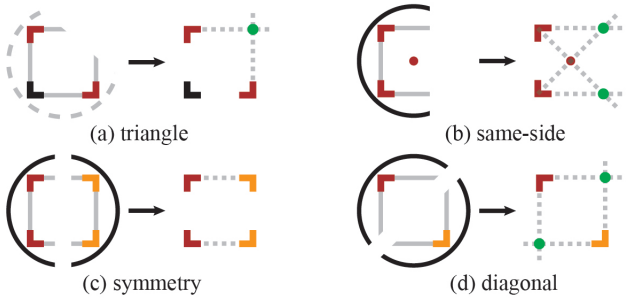


Fig. 5. The four common occlusion scenarios and their corresponding recovery methods. The dashed ellipse in Figure(a) signifies that its presence is not mandatory. The corners highlighted in orange and red denote those involved in the recovery process, with distinct colors indicating their association with different edge segments. The green point represents the estimated corner position. The red dot in Figure(b) serves as the ellipse center, which can also be regarded as the marker center

C. Encoding and Decoding

CopperTag utilizes the Reed-Solomon (RS) [22] algorithm for encoding and decoding. An RS code is represented as $RS(n,k)$ with s-bit symbols, where n denotes the total number of symbols in the codeword, and k represents the count of data symbols. The codeword consists of $n-k$ parity symbols and can correct up to a maximum of $(n-k)/2$ error symbols. Specifically, CopperTag-8 employs $RS(15,5)$ with 4-bit symbols, while CopperTag-14 uses $RS(20,4)$ with 8-bit symbols, allowing for diverse symbol sets, like letters. Figure 3(b) exemplifies a CopperTag-14 with content "abc".

During the decoding process, CopperTag systematically samples bit values in a predefined order within the quadrilateral region generated by the preceding processing step. In the case of CopperTag-8, sampling begins in region 1 and proceeds sequentially to region 15, as shown in Figure 3(a). Within each region, the sampling sequence follows a left-to-right and top-to-bottom pattern.

IV. Experiment

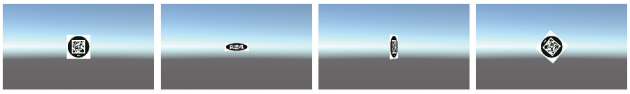
A. Simulation Experiment

We conducted both simulation and physical experiments to evaluate the proposed CopperTag system. In the simulation experiment, we created a virtual camera within Unity3D to capture markers in various positions, generating a dataset for analysis[2]. Four different markers were employed for comparison: ArUco3, AprilTag3, STag and CopperTag. Each marker image was set to a size of 0.1mx0.1m, while the virtual camera was configured with a 16mm focal length and a 1920x1080 imaging capability.

The simulation dataset consisted of four subsets for each marker, denoted as “distance-z”, “rotated-x”, “rotated-y” and “rotated-z”, as illustrated in Figure 6. In all these

[2]The complete datasets is available in the code repository

sets, the marker was positioned in front of the camera. In the "distance-z" dataset, the marker gradually moved away from the camera in 0.1m increments, ranging from 0.3m to 2.8m, resulting in 251 images. The "rotated-x" and "rotated-y" datasets represented rotations around the x or y axis, spanning from -70 degrees to 70 degrees with a 1-degree resolution, yielding 141 images for each set. The "rotated-z" dataset depicted rotations around the z-axis from 0 degrees to 360 degrees, comprising 361 images.



(a) distance-z (b) rotated-x (c) rotated-y (d) rotated-z

Fig. 6. The examples from the simulation dataset of CopperTag. In Figure(a), CopperTag is positioned 1 meter in front of the camera. In Figure(b), CopperTag is rotated -70 degrees around the x-axis, and Figure(c), it is rotated 70 degrees around the y-axis. In Figure(d), CopperTag is rotated 270 degrees around the z-axis

Our experimental platform utilized a laptop with an Intel i7-10750H CPU operating at 2.6GHz, featuring 8 cores and 8GB of memory, running Ubuntu 20.04. We assessed marker detection rates, processing times, and computational resource utilization across this simulation dataset, presenting the results in Table I. The detection rate, defined as the number of successfully detected images divided by the total number of images, reached 100% for ArUco3, AprilTag3 and CopperTag. However, STag had 50 images with failed detection in the "distance-z" subset. The "Time" column in Table I represents the average processing time for each marker within its respective simulation dataset. CopperTag exhibited the fastest performance at 4.41ms, followed by ArUco3, AprilTag3 and STag. Since all four markers employed a single-threaded approach, their CPU usage was similar, utilizing a full load of a single core, equivalent to 13.5%. The differences in memory usage among these markers were negligible, with a gap of approximately 164MB between maximum and minimum usage.

TABLE I. The comparison of detection rate, speed and resource usage among four fiducial markers system in a simulation dataset

Fiducial Marker	Performance		Resource (%)	
	Detection Rate	Time	CPU	memory
ArUco3	100%	5.44ms	14.07	1.27
AprilTag3	100%	12.23ms	13.47	2.9
STag	94.4%	23.11ms	13.64	0.93
CopperTag	100%	4.41ms	13.24	2.65

We also evaluated the absolute error of each corner pixel in every simulation image. The ground truth of the corner pixel location was determined based on the original corner pixel location and the current relative pose between the marker and the camera, provided by Unity3D. The results for these four sets are presented in Figure 7. The trends and ranges of these four markers were similar, with STag exhibiting the largest pixel error variation. In Figure 7(a), the STag line is discontinuous because it failed to detect at that distance, rendering it unable to calculate the absolute pixel error. The possible reason for STag detection failure is its sensitivity to parameters. In this test, we only utilized default parameters rather than the tuned parameters, which may have contributed to the failure.

One of CopperTag's significant contributions lies in its ability to detect markers under occlusion conditions. To evaluate its performance, we created four occlusion levels: 5%, 10%, 15% and 20%. Each level represented the extent of marker area destruction. Each level included five images, resulting in a total 20 images in the occlusion dataset, as shown in Figure 8. This dataset encompassed not only the triangle, same-side, symmetry, and the diagonal situations mentioned in Figure 5 but also inner information area occlusion scenarios. We overlaid these 20 occlusion patterns onto ArUco3, AprilTag3, STag, and CopperTag to create occluded images and tested their detection ability. The results are presented in Table II. ArUco3 consistently failed in all occlusion patterns due to the destruction of its convex quadrilateral assumption. AprilTag3 only detected markers when the occlusion patterns did not cover the inner square. STag managed to detect 12 images among the occlusion patterns but could not handle same-side, diagonal, and symmetry conditions. In contrast, CopperTag successfully detected all occlusion patterns.

B. Physical Experiment

We conducted physical experiments using an industrial camera, MV-CE050-30GM, equipped with a 3.5mm focal length lens to capture a 0.05mx0.05m CopperTag at a resolution of 1920x1080, as shown in Figure 9. The computational platform remained the same as in the simulation experiment.

Table III presents the detection rates for the physical dataset. The "usual" dataset achieved a 100% detection rate, while the "challenge" dataset only reached a 48% detection rate, primarily due to the inability to extract precise edge segments from defocused images, resulting in noisy edge segments that formed invalid quadrilaterals, leading to detection failures. The detection rate in the physical occlusion dataset decreased with increasing occlusion levels. Compared to canonical occlusion images in Figure 8, physical imaging introduced additional noise from the camera system, ambient lighting, and the pose between CopperTag and the camera. Most of the failed detection in occlusion images were due to small size, excessive slanting, or pollution of more than two corners. If an incomplete CopperTag was too distant from the camera, its size in the image would be too small for detection; thus, we do not recommend using CopperTags smaller than 48x48 pixels for detection. In situations with excessive slanting, while the AAMED method could still identify a similar ellipse and its center, it did not accurately represent the marker center. Consequently, missing corners could not be correctly recovered.

We also recorded the processing time for each image in the physical dataset, as shown in the second row of Table III. Processing times for physical images were significantly higher than for simulation images due to the complex backgrounds in physical images, resulting in more edge segments.

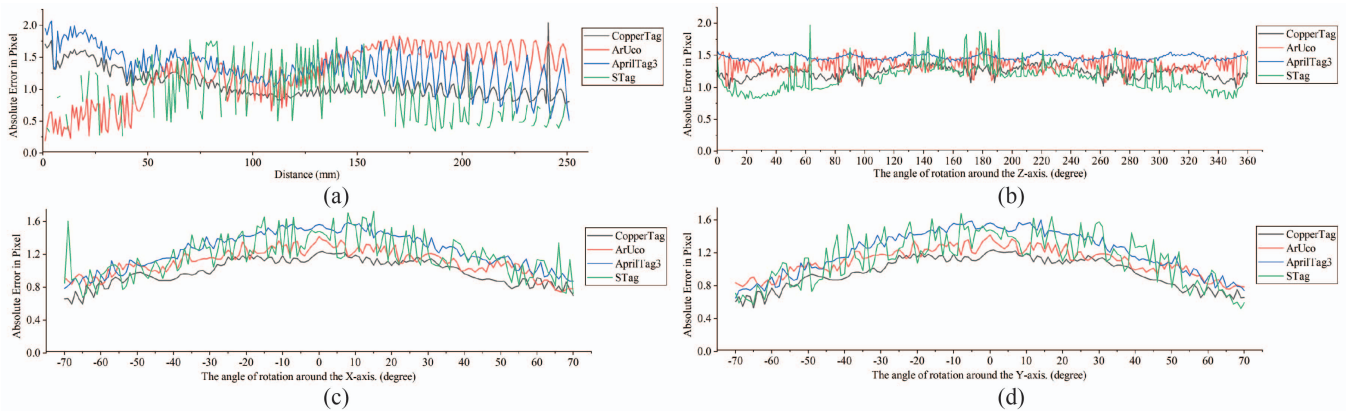
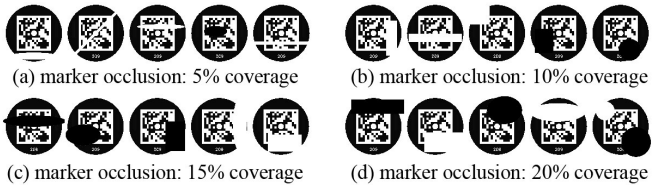


Fig. 7. The absolute error of the corner pixel varies with changes in the distance or angle between the camera and the marker. Figure (a) represents the results from the "distance-z" dataset, while Figures (b), (c), and (d) represent the results from the "rotate-z", "rotate-x", and "rotate-y" datasets, respectively. The vertical axis represents the absolute error in pixels of the current image, while the horizontal axis in each subplot represents different images within the same dataset.

TABLE II. The Detection of four fiducial markers in the presence of four progressively occluded scenarios

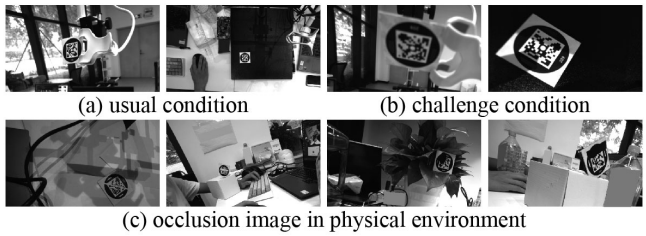
Fiducial Marker	5%					10%					15%					20%				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
ArUco3																				
AprilTag3	✓			✓																
STag	✓		✓	✓		✓		✓	✓	✓		✓	✓		✓	✓		✓		
CopperTag	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓



(a) marker occlusion: 5% coverage (b) marker occlusion: 10% coverage

(c) marker occlusion: 15% coverage (d) marker occlusion: 20% coverage

Fig. 8. The four progressively occluded scenarios are illustrated in Figures (a), (b), (c), and (d), representing the generated image groups with 5%, 10%, 15%, and 20% area coverage, respectively. The four image groups contain the previously mentioned occlusion scenarios: triangle, same-side, symmetry, and diagonal



(a) usual condition (b) challenge condition

(c) occlusion image in physical environment

Fig. 9. The images from the CopperTag physical dataset are depicted. Figure (a) represents the single intact CopperTag in a typical indoor environment. Figure (b) represents the single intact CopperTag under challenging conditions, such as out of focus and uneven lighting. Figure (c) represents the occluded CopperTags under both typical and challenging conditions

TABLE III. The performance of CopperTag in various physical conditions

	I_s[a]	I_c[b]	O_{5p}	O_{10p}	O_{15p}	O_{20p}[c]
Detection Rate	100%	48%	83.5%	80.5%	67%	63%
Processing Time (ms)	7.39	6.09	9.27	9.39	9.44	9.15

[a] I_s: single intact CopperTag in usual environment
[b] I_c: single intact CopperTag in challenging environment
[c] O_{xp}: single CopperTag occluded by x% area in physical environment

Processing times for occlusion markers were greater than those for intact markers, as they required more time to search for candidate CopperTag regions, as indicated in Algorithm 2. Nevertheless, the processing times remained within acceptable limits for most real-time robotics applications.

V. Conclusions

In this paper, we present CopperTag, a novel fiducial marker specifically engineered to improve detection robustness in robotics applications, especially in scenarios involving occlusions. Our experiments demonstrate that CopperTag exhibits detection performance comparable to commonly employed markers like ArUco3 and AprilTag3 while significantly outperforming them in occluded situations. With an average processing time of 10ms per frame, CopperTag meets the stringent real-time demands of industrial robotics tasks.

Looking ahead, CopperTag still has some disadvantages that need to be addressed, such as detection at far distances. This is because the line and corner features can easily deteriorate when CopperTag appears too small in the image, and the inner information area may become too small for decoding. Our future research will concentrate on improving the detection rate at far distances and refining the encoding and decoding rules. We aim to explore the integration of other marker dictionaries into CopperTag to enhance its versatility and adaptability.

REFERENCES

[1] M. Fiala, "Designing highly reliable fiducial markers," *IEEE Transactions on Pattern analysis and machine intelligence*, vol. 32, no. 7, pp. 1317–1324, 2009.

[2] A. Babinec, L. Jurišica, P. Hubinský, and F. Duchoň, "Visual localization of mobile robot using artificial markers," *Procedia Engineering*, vol. 96, pp. 1–9, 2014, modelling of Mechanical and Mechatronic Systems.

[3] B. Mantha and B. Garcia de Soto, "Designing a reliable fiducial marker network for autonomous indoor robot navigation," in *Proceedings of the 36th International Symposium on Automation and Robotics in Construction (ISARC)*, 2019, pp. 74–81.

[4] F. Guangrui and W. Geng, "Vision-based autonomous docking and re-charging system for mobile robot in warehouse environment," in *2017 2nd International Conference on Robotics and Automation Engineering (ICRAE)*. IEEE, 2017, pp. 79–83.

[5] A. Annusewicz-Mistal, D. S. Pietrala, P. A. Laski, J. Zwierzchowski, K. Borkowski, G. Bracha, K. Borycki, S. Kostecki, and D. Wlodarczyk, "Autonomous manipulator of a mobile robot based on a vision system," *Applied Sciences*, vol. 13, no. 1, 2023.

[6] L. F. Belussi and N. S. Hirata, "Fast component-based qr code detection in arbitrarily acquired images," *Journal of mathematical imaging and vision*, vol. 45, pp. 277–292, 2013.

[7] L. Karrach and E. Pivarčiová, "Comparative study of data matrix codes localization and recognition methods," *Journal of Imaging*, vol. 7, no. 9, p. 163, 2021.

[8] H. Kato and M. Billinghurst, "Marker tracking and hmd calibration for a video-based augmented reality conferencing system," in *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*. IEEE, 1999, pp. 85–94.

[9] M. Fiala, "Designing highly reliable fiducial markers," *IEEE Transactions on Pattern analysis and machine intelligence*, vol. 32, no. 7, pp. 1317–1324, 2009.

[10] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE International conference on robotics and automation*. IEEE, 2011, pp. 3400–3407.

[11] J. Wang and 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.

[12] M. Krogius, A. Haggenmiller, and E. Olson, "Flexible layouts for fiducial tags," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 1898–1903.

[13] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014.

[14] F. J. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, "Speeded up detection of squared fiducial markers," *Image and vision Computing*, vol. 76, pp. 38–47, 2018.

[15] B. Benligiray, C. Topal, and C. Akinlar, "Stag: A stable fiducial marker system," *Image and Vision Computing*, vol. 89, pp. 158–169, 2019.

[16] F. Bergamasco, A. Albarelli, E. Rodola, and A. Torsello, "Rune-tag: A high accuracy fiducial marker with strong occlusion resilience," in *CVPR 2011*. IEEE, 2011, pp. 113–120.

[17] L. Calvet, P. Gurdjos, C. Griwodz, and S. Gasparini, "Detection and accurate localization of circular fiducials under highly challenging conditions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 562–570.

[18] H. Wang, Z. Shi, G. Lu, and Y. Zhong, "Hierarchical fiducial marker design for pose estimation in large-scale scenarios," *Journal of Field Robotics*, vol. 35, no. 6, pp. 835–849, 2018.

[19] F. J. Romero-Ramire, R. Munoz-Salinas, and R. Medina-Carnicer, "Fractal markers: a new approach for long-range marker pose estimation under occlusion," *IEEE Access*, vol. 7, pp. 169 908–169 919, 2019.

[20] S. Suzuki and K. be, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, 1985.

[21] C. Meng, Z. Li, X. Bai, and F. Zhou, "Arc adjacency matrix-based fast ellipse detection," *IEEE Trans. Image Process.*, vol. 29, p. 4406–4420, 2020.

[22] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.