Quadcopter Tracks Quadcopter via Real Time Shape Fitting

Dror Epstein[1] and Dan Feldman[2*]

Abstract

We suggest a novel algorithm that tracks given shapes in real-time from a low quality video stream. The algorithm is based on a careful selection of a small subset of pixels, that suffices to obtain an approximation of the observed shape. The shape can then be extracted quickly from the small subset. We implemented the algorithm in a system for mutual localization of a group of low-cost toy-quadcopters. Each quadcopter carries only a single 8-gram RGB camera, and stabilizes itself via real-time tracking of the other quadcopters in ~ 30 *frames per second.*

Existing algorithms for real-time shape fitting are based on more expensive hardware, external cameras, or have significantly worse performance. We provide full open source to our algorithm, experimental results, benchmarks, and video that demonstrates our system. We then discuss generalizations to other shapes, and extensions for more robotics applications.

1. Introduction

Motivation. The problem of shape fitting into a set of points is fundamental in computer vision and image processing, with many applications in robotics. It became an integral part of our everyday life, including: Safety activities such as tracking objects or people [1,2], recognition of barriers and obstacles for autonomous vehicles [3,4], identification of markers for guiding and navigation implementation by robots [5, 6], automatic inspection of manufactured products [7], eye tracking for robot-human communication [8,9] and many more. These systems raise the needs of a robust, accurate and very fast algorithms for shape fitting, which is the recognition of known geometric shapes in an image.

While there are many potential applications for real-time shape fitting, this paper was specifically motivated by the application of mutual localization in a swarm of toy quadcopters, which require update rate of about 30 FPS (frames per second).

In recent years, such toy micro quadcopters are extremely common and easy to purchase in supermarkets and on-line stores such as Amazon. Moreover, they are usually legal, relatively low-cost ($20-$50), and safe for indoor navigation. However, the lack of sensors makes them extremely unstable, and requires ~ 30 remote controller commands per second for a stable hovering, which is impossible for a human.

Most of the existing solutions for control and navigation of vehicles are based on external sensors such as cameras, GPS and radars. However, for autonomous vehicles such as toy-quadcopters, these solutions are expensive, heavy, large and demand some mechanical stabilization mechanism. Sensors such as GPS cannot be carried by such quadcopters, but even for larger quadcopters, we aim for an error of at most few centimeters, and indoor robots that lack GPS support.

Example applications and IoT. Our main motivation was mutual real-time stabilization, but we expect that our open system will be used and extended for many other real-time applications for robots and devices that need tracking algorithms that are based on or want to use a single weak low-weight noisy RGB-camera. Examples include real-time tracking of people or objects, visual SLAM (simultaneous localization and mapping), autonomous cars or crawler robots, for indoor operation, GPS denied environment and many more. In particular, IoT (Internet of Things) applications usually run on such weak, lightweight, and low-cost devices and mini-computers that e.g. can be carried by toy drones or are wearable by a human; see survey of such applications in [10]. For example, the 8-gram camera that is used by our system with its tiny battery can be used as a wearable device or part of it, and our algorithm may be extended for real-time tracking, e.g., for sports activities. Our quadcopters are also controlled by a mini-computer that is common in IoT applications. See "Arduino" in the hardware section.

While there are many other solutions for these applications, our algorithm specialized in real-time shape fitting with performance and quality that we were not able to achieve with existing results for our following application.

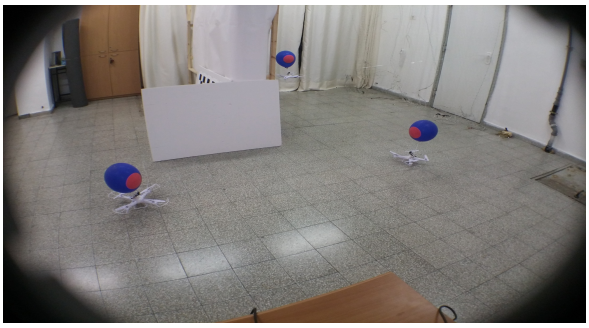
Localization of quadcopters via quadcopters. We demonstrate our algorithm with a system for localiza-

[*1] Dror Epstein and Dan Feldman are with the Robotic & Big Data Lab at the Department of Computer Science, University of Haifa, Israel dror.epstein,dannyf.gmail.com

tion of a swarm of quadcopters, where each quadcopter carries only a (monocular) low-weight (few grams) camera. To avoid collision, maintain formation, or just to hover in the sky when there are no visual markers around, each quadcopter may localize itself by using only other surrounding quadcopters. This is maybe similar to flocking behavior of birds, where each bird positions itself based on other birds. To simplify the detection, we attach a balloon to each quadcopter; see Fig. 1 and our video [11]. This system, as the other applications above, raises the following problem of robust real-time shape fitting.

The localization problem is then reduced to the following visual pattern recognition problem.

Figure 1: *A snapshot from the experiment in our lab, where quadcopters hover based on mutual tracking of other quadcopters. The leftmost quadcopter positions itself based on the middle quadcopter, which in turn localizes itself via the last quadcopter which is assumed to be relatively stable.*



Problem statement. The basic problem definition of *Circle Detection* is to compute a circle that best fits a collection of pixels in an image and reduces the sum of distances between the circle to each of the points in the collection. Moreover, we aim to detect such a moving circle in a real-time video. The algorithm thus gets as input a real-time video stream at frame rate of up to roughly 500 FPS. Each image is assumed to contain a circle, but also to be noisy, unstable, and to have an unknown background. The output of the algorithm is a stream of estimated position of circles, where each circle (radius and center) corresponds to a frame in the input stream. the main challenge is to handle these output rates but still obtain the desired accuracy.

Novel technique: real-time match-set. Instead of trying to develop a new algorithm from scratch, our approach is to first carefully select a very small set of pixels in the image, called *matched-set*. This set represents the original (full) set of pixels in the sense that running a simple and fast algorithm on the matched set yields a result (approximated shape) that is similar to the resulting circle that we get by running the algorithm on the original image (full set of pixels). Furthermore, an advantage of our algorithm is that we can extract the shape even if parts of the shape are missing or obscured. Since the main algorithm is run only on the matched-set, our main algorithm has running time that is *sub-linear* in the input size.

In theory, the cost for using this data summarization approach is that we get only an approximated solution to the desired shape. In practice, the resulting shape is even *more* similar to the ground truth, i.e., the shape in the real-world, since the matched set also avoids the selection of outliers and noisy pixels. The main challenge in this paper is to compute such a matched set that we can (a) track in real-time, and (b) get sufficiently accurate and fast approximation for our application.

Another common disadvantage of existing algorithms such as [12, 13] is that they are unstable: the output circles are too far from each other in each iteration. Our algorithm outputs in each frame a circle that is close to the previous detected circle, so that the position and size of the detected circles are relatively stable and consistent.

Generalizations and extension to other shapes. For simplification, we present our algorithm and the matched-set paradigm for detecting circles. However, our algorithm is relatively generic and can easily be adopted for other convex shapes. More precisely, there are two requirements for our algorithm: (1) each shape in the desired family of shapes has small complexity in the sense that it can be described by a small number of parameters. This size of the matched-set is approximately linear with this number, and (2) a "black box" algorithm that fits a non-noisy small matched set (subset) to a given image.

Our main contributions are as follows.

1. An algorithm for real-time circle detection in a streaming video of a rate of up to 500 FPS.
2. Experimental results on both synthetic and real data, as well as comparison to state of the art algorithms.
3. A system for mutual tracking of toy quadcopters using our algorithm. Each quadcopter carries only an 8-gram monocular RGB camera and a balloon.

Related work. Detection of circular objects in digital images is an important and recurring problem in image processing [14] and computer vision [15]. In [16], the authors describe the process and the challenge of shape fitting such as unsterile environment, background noise and outliers. Most of the circle detection algorithms

are based on the well known *Circle Hough transform* (CHT) [17, 18], or geometric characteristics. The purpose of these techniques is to find circles in imperfect image inputs. The circle candidates are produced by voting in the Hough parameter space and then select the local maxima in a so-called accumulator matrix. The main drawback of these techniques is their running time complexity which is usually linear while sub-linear time algorithms are needed. Moreover, since we deal with corrupted images, the CHT technique is able to detect different parts of circle from different area of the balloon for each image, and therefore the algorithm outputs a circle that moves in a non-continuous form (circles in consecutive frames may be very far from each other).

Cuevas et al. [19] present Learning Automata (LA), which is a heuristic method to solve complex multi-modal optimization problems. The detection process is considered as a multi-modal optimization problem, allowing the detection of multiple circular shapes through only one optimization procedure.

Akinlar and Topal [13] present an algorithm that makes use of the contiguous (connected) set of edge segments produced by there edge drawing parameter free (EDPF) algorithm. They suggest an algorithm with time complexity of $O(n^2)$ for such an image.

In [20], Zhou et al. present a new circular object detection method based on geometric property and polynomial fitting in polar coordinates instead of implementing it in Cartesian coordinates. It is tailored for 2-Dimension ($2D$) LIDAR data.

Zhang, Wiklund and Andersson [21] provide a circle detection algorithm based on randomized sampling of isosceles triangles (ITs), that provides distinctive probability distribution for circular shapes with a low amount of iterations. Although they introduced an accurate and robust detection at sharp images, it does not handle corrupted and moving images.

Zhou and He [22] propose a modified version of Hough Transform, called Vector Quantization (VQHT), to detect circles more efficiently, by decompose the edges in the image into many subimages, then, the edge points resided in each sub-image are considered as one circle candidate group.

In [23], the authors introduce *ChromaTag*, a colored fiducial marker and detection algorithm that accomplish detection within 1.4 millisecond.

Overview. In Section 2 we present and describe our circle detection algorithm. Section 3 explains how to obtain localization from the streaming circles using few formulas. Section 4 presents our real-time system for autonomous group of toy-quadcopters. Section 5 shows experimental results using our algorithm, and measures its accuracy and running time compared to other algorithms. Section 6 concludes our work.

2. Circle Detection Algorithm

In this section we present our main circle detection algorithm. It aims to quickly detect a circle while overcoming noise that might occur due to hidden objects, image corruptions, light distortion, distorted shapes and other types of noise that are common in a video stream from an RGB camera that is mounted on an unstable device such as a hovering quadcopter.

The algorithm first identifies a small subset of points, called *matched-set*, which represents an existing circle. Then, it matches each point in this match-set to a point in the new image. Finally, it extracts and output the new circle from the "clean" matched set in the new image, based on a naive and fast algorithm.

Notation. The set $\{1,\cdots,n\}$ is denoted by $[n]$. An *image* $M \in \{0,1\}^{n\times n}$ is a binary matrix. For every $(x,y,r) \in \mathbb{R}^3$, we denote by $C(x,y,r)$ the circle of radius r on the plane that is centered at $p_C := (x,y)$. for convenience we define $C(x,y,r) := C(p_c,r)$. A point $(x,y) \in [n]^2$ is *white* if the (x,y)-entry of M is 1, and *black* otherwise.

The points on the boundary of the fitted circle are called edge-points. The expected width of the circle (its margin) is denoted by $\partial > 0$. We denote by $\nabla > 0$ the threshold that defines the required difference between black area and white area. These parameters are defined based on the expected type of noise and robustness of our algorithm. Larger margins and high threshold will allow the algorithm to be more robust to outliers, noise, blurring etc. but will result in a larger matched set and thus slower computation time. Smaller margins and threshold assume less noise and will result in a smaller and accurate matched-set.

Each edge is also associated with its orientation or direction $dir \in directions = \{right, left, upper, lower\}$.

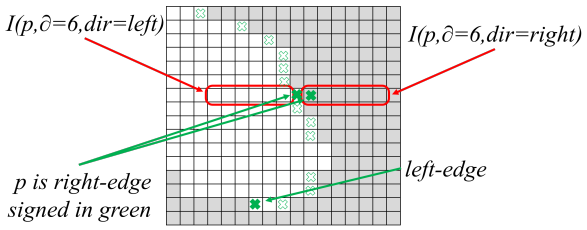
Formally, for a point $p = (x,y) \in [n]^2$, we define $I(p,\partial,dir)$ to be the set of white points in M of distance at most ∂ in the direction dir from $p = (x,y)$ as follows.

$$I(p,\partial,dir) = \{\ (x',y') \in [n]^2 \mid M(x',y') = 1, \text{ and } \begin{cases} x' \in [x,x+\partial], y = y' & \text{if } dir = right, \\ x' \in [x-\partial,x], y = y' & \text{if } dir = left, \\ x = x', y' \in [y-\partial,y] & \text{if } dir = upper, \\ x = x', y' \in [y,y+\partial] & \text{if } dir = lower \end{cases} \}. \quad (1)$$

For $\nabla > 0$ and $\partial > 0$, the point p is called *right-edge* if $|I(p,\partial,left)| - |I(p,\partial,right)| > \nabla$ and we denote the union of all right-edges in M over $p \in [n]^2$ by $E(M)^{right}$. Similarly, in a symmetric way, we denote by $E(M)^{left}$, $E(M)^{upper}$ and $E(M)^{lower}$ respectively the union sets of the *left-edges*, *upper-edges* and

lower-edges. Their union of edge points is $E(M) = \{E(M)^{right} \cup E(M)^{left} \cup E(M)^{upper} \cup E(M)^{lower}\}$. An example for right-edge points with the parameters $\nabla = 0.1$ and $\partial = 6$ is shown in Fig. 2.

Figure 2: *Illustration for edge points detection.*



For a circle $C(x,y,r)$, an edge-points set $E(M)$ and a threshold $\varepsilon \in (0,1)$, we denote by $E(M,C)$ the edge-points which are approximately on the circle, i.e., of distance $r \pm \varepsilon$ from the center (x,y). In Fig. 3a the set $E(M,C)$ is marked in black. Formally,

$$E(M,C) = \left\{ \; e \in E(M) \mid \|e-(x,y)\| \in [r-\varepsilon, r+\varepsilon] \; \right\}. \tag{2}$$

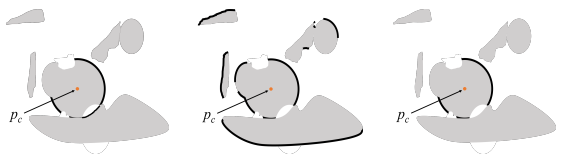
We define the set of *arc-segments* for a point p as the collection of edge-points such that their black area side is closer to point p then the white area. In Fig. 3b the set $A(M,p)$ is marked in black. Formally,

$$A(M,p) = \left\{ e \;\middle|\; \begin{array}{l} e \in \{E(M)^{left} \cup E(M)^{lower}\} \\ \text{if } p \text{ is above and right to point } e, \\ e \in \{E(M)^{right} \cup E(M)^{lower}\} \\ \text{if } p \text{ is above and left to point } e, \\ e \in \{E(M)^{left} \cup E(M)^{upper}\} \\ \text{if } p \text{ is below and right to point } e, \\ e \in \{E(M)^{right} \cup E(M)^{upper}\} \\ \text{if } p \text{ is below and left to point } e \end{array} \right\}. \tag{3}$$

Let C be a circle, and P_c denotes its center. The *match-set* of C is the union of edge-points such that each edge-point is also in an arc segments of the circle C. In Fig. 3c the match-set $S(M,C)$ is marked in black. Formally,

$$S(M,C) = \{E(M,C) \cap A(M,p_C)\}. \tag{4}$$

Figure 3: *(a) $E(M,C)$ - An edge-points set that lies on the boundary of the circle C. (b) $A(M,p)$ - arc segments set of the point p. (c) $S(M,C)$ - match-set of C.*



(a) (b) (c)

Let $s_1, s_2, s_3 \in S(M,C)$ represents three match points of the circle match-set. We denote by CENTER(s_1,s_2,s_3) the center point of the circumscribed circle related to points s_1, s_2 and s_3. Let T be a set of points. We denote by AVERAGECENTER(T) the middle point (x,y) of all points in T. We denote by DISTANCE(p_a,p_b) the *Euclidean distance* between points p_a and p_b. We denote by CLOSESTPOINT(p,A) the closest point from the set A to a point p.

Overview of Algorithm 1: circle detection. The circle detection algorithm, as shown in Algorithm 1, continuously detects circles over a video stream. The detection is based on the matched-set as defined in (4), which represents the edge-points on the detected circle that is computed in Algorithm 2. For each frame we then apply Algorithm 3 that tracks those edge-points in the new frame to extract the desired circle by a naive algorithm that estimates a circle from non-noisy data. In Algorithm 1, the ith iteration corresponds to the ith image, for every $i \geq 1$.

Algorithm 1: CIRCLEDETECTION

Input: A stream of images $M_0, M_1 \cdots \in [n]^2$.
Output: A stream of circles $C_0, C_1, \ldots \in \mathbb{R}^2$.

```
/* Circle initialization                    */
```
1 Set $(C_0, S) \leftarrow$ INITCIRCLE(M_0)
2 **Output** C_0
```
/* Iterative tracking over
   stream frames.                           */
```
3 **for** $i \leftarrow 0$ *to* ∞ **do**
4 | Set $(C_i, S) \leftarrow$ TRACKCIRCLE$((M_i, S, C_{i-1}))$
5 | **Output** C_i

Overview of Algorithm 2: initialization. In Line 1 we identify all the edge-points in the image as defined in (2); see Fig. 4b. In Lines 2– 5, we extract a circle in a way that is similar to the CHT algorithm; see Fig. 4c. In Line 6, we identify the circle match-set union; see Fig. 4d.

Algorithm 2: INITCIRCLE(M)

Input: An image $M \in [n]^2$.
Output: A circle C', and a match-set S'.

```
/* Detect edges; See Fig.   4b.
   */
```
1 Set $E \leftarrow$ union of edge-points in M ; See (2)
2 Set $C' \leftarrow C(0,0,0)$
```
// Identify circle; See Fig. 4c.
```
3 **for** ***each*** $(x,y,r) \in [n]^3$ **do**
4 | **if** $|S(M,C(x,y,r))| > |S(M,C')|$ **then**
5 | | Set $C' \leftarrow C(x,y,r)$
```
/* Compute match-set.                       */
```
6 Set $S' \leftarrow S(M,C')$; See (4)
7 **Output** C', S'

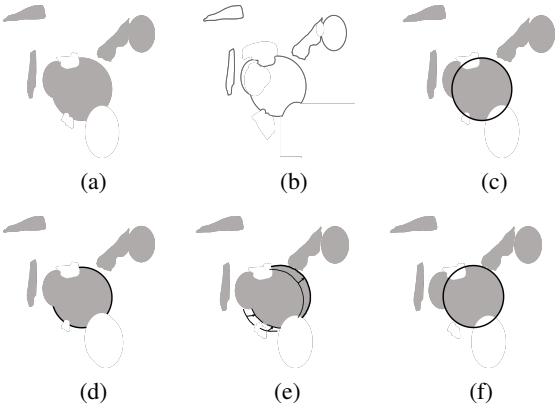
Overview of Algorithm 3: tracking. In Line 1 we fit each point on the circle match-set to one of the edge-points; see Fig. 4e. The fitting technique ensures that each of the selected edge-points represents a real segment arc that belongs to the new circle. This enables us, in Lines 3– 6, to approximate an accurate new circle via analytic solution, while using some technique to eliminate outliers and improve our detection; see Fig. 4f. In Line 7, we update the match-set according to the new detected circle for the next iteration; see Fig. 4d. Finally, in Line 8, we output the detected circle. For simplicity, we removed validation parts from the algorithm e.g., division by 0.

Algorithm 3: TRACKCIRCLE(M, H, R)

Input: An image $M \in [n]^2$, a match-set H, and a circle R.
Output: A circle C' and a match-set S'.

Set $H' \leftarrow \emptyset$
`/* Fit match-set; See Fig. 4e */`
1 **for** ***each*** $h \in H$ **do**
2 | Set $H' \leftarrow H' \cup \{ClosestPoint(h, A(M, p_R)\}$ See (3)
`// Calculate circle; See Fig. 4f`
3 Set $T = \{Center(h'_{3i}, h'_{3i+1}, h'_{3i+2}) \mid h' \in H', i \in 0, 1, 2, ..., \frac{n}{3}\}$
4 Set $p_c \leftarrow AverageCenter(T)$
5 Set $r \leftarrow \frac{\sum_{h' \in H'} Distance(p_c, h')}{|H'|}$
6 Set $C' \leftarrow C(p_c, r)$
`// Update match-set; See Fig. 4d`
7 Set $S' \leftarrow \{h' \in H' \mid \|h' - p_c\| \in [r-1, r+1]\}$
8 **Output** (C', S')

Figure 4: *Procedure snapshots. (a) Input image; (b) Detect edges; (c) Identify circle; (d) Identify match-set; (e) Fit match-set; (f) Calculate circle.*



Run-time analysis. Algorithm 1 includes an initialization part; see Algorithm 2, and iterative tracking part; see Algorithm 3. Hence, the running time is dominated by the tracking algorithm. The running time of the tracking algorithm is the sum over the time for the match-set fitting, line 1, circle calculating, line 3, and the match-set updating, line 7, that run on a small subset of points (match-set). The matche set is of size $O(n)$ since this is the size (number of pixels) of a circle in an $n \times n$ grid of pixels. Hence, the total running time complexity of the algorithm (excluding initialization) is $O(n)$, i.e., sub-linear in the input image size $O(n^2)$.

Table 1: Running time per frame in a streaming video.

Algorithm	Detection time (millisecond)
EDCircles [13]	9.9
GRCD-R [24]	52
Isophote [25]	20.5
ITCiD [21]	15.7
PolynomialFitting [20]	16.9
AR Aristo platform [26]	16.6
RuneTag [27]	51
AprilTag [28]	19
ChromaTag [23]	1.4
Algorithm 1	0.9

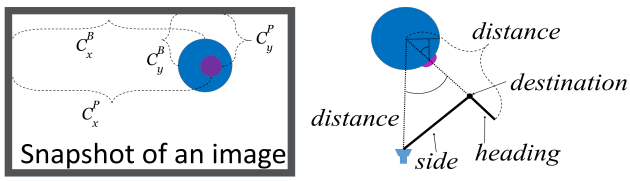
Table 1 compares the actual running time of our algorithm with existing solutions.

3. Localization via Circle Fitting

Localization of an object based on other two known objects, i.e, 6 degrees of freedoms (6DOF, position and orientation) can be computed via the following simple formulas. The approach is based on a visual perception technique, *parallax* [29], which computes the position of an object based on the appearance in the image of the two observed objects (size and position).

The two referenced objects may be represented by a ball (or its center) and a marker on the shell of the ball; see Fig. 5 and Eq. (5)– (9). These equations are based on the input parameters BR (the radius of the ball), β (the camera's field of view) and n (the number of pixels per row). The rest of the parameters are the circle solutions that computed by Algorithm 1, denoted by C^B and C^P, respectively, for the blue ball and the purple marker circles. For convenience, we denote by C_x, C_y and C_r the circle parameters.

Figure 5: *Geometric scheme for relative position.*



$$distance = \frac{\frac{n}{2} \cdot BR}{C_r^B \cdot \tan\left(\frac{\beta}{2}\right)} \quad (5)$$

$$side = \frac{\pi^2 \cdot distance \cdot \arcsin\left(\frac{C_x^P - C_x^B}{C_r^B}\right)}{2} \quad (6)$$

$$height = \frac{\pi^2 \cdot distance \cdot \arcsin\left(\frac{C_y^P - C_y^B}{C_r^B}\right)}{2} \quad (7)$$

$$heading = \sqrt{distance^2 - side^2 - height^2} \quad (8)$$

$$yaw = \arctan\left(\frac{C_x^B - \frac{n}{2}}{distance}\right) \quad (9)$$

4. System:Quadcopter Tracks Quadcopter

We implemented our algorithm and then designed and built a system to evaluate its performance for our application. The goal is to allow each quadcopter in a swarm of quadcopters to localize itself, via a single monocular camera and a marked balloon that is attached to each quadcopter. Unlike other papers, we assume that there are no external visual markers except the other quadcopters. Each quadcopter tries to keep a stable hovering solely based on the other quadcopters, independently of the background, or environment.

The resulting 6DOF of each quadcopter are being sent to its controller. The controller computes the desired corrections to fix the quadcopter in place. The resulting commands are then sent to the remote controller of the quadcopter. A rate of roughly 30 FPS is needed to keep the quadcopter in place.

Our system suggests the first step toward such an autonomous swarm of quadcopters, based on Algorithm 1.

The setup. The system consists of three quadcopters. The first quadcopter is equipped with a tiny camera that is pointed toward the second quadcopter. The first quadcopter positions itself in real-time only using the second quadcopter. Similarly, the camera on the second quadcopter is pointed toward the third quadcopter, which is used to localize the second quadcopter. Finally, the third quacopter is hanged by a swinging rope to the ceiling.

Hardware. The following is a list of the used hardware in our system.

3**x Toy quadcopters**. Each quadcopter has no positioning sensors, weigh only 100 grams, costs less than $50, and thus relatively safe and low-cost but unstable. See [30].

2**x Monecular camera**. Two out of the three quadcopters are equipped with a monocular RGB analog camera, and weigh only 8 grams, including internal video transmitter. It transmits 30 RGB FPS in PAL coding and resolution of 640×480 pixels. See [31].

2**x Video receivers**. Analog video receivers, each receives data from a different channel of a different TX camera.

2**x Analog to Digital converter**. This converter is connected to the video receiver and converts its analog video to a digital video that is being sent to the laptop below via USB 2.0 connection; see [32].

2**x Laptops**. The streaming input video is processed by Algorithm 1 that is run on a standard laptop. The output is a stream of commands that are sent to an Arduino board below. We use *Intel*® $4810MQ, 2.8$ GHz CPU. See [33].

2**x Arduino mini-boards**. Each board gets a stream of serial I/O commands from the laptop and sends them to the remote controller of the quadcopter. See [34].

2**x Remote controllers**. These controllers are shipped with the SYMA quadcopters. We reverse engineered their protocols and therefore can send commands from the Arduino boards, instead of a human, in about 30 FPS.

2**x balloons**. Each balloon is blue with a purple small ball marker in its center. It is carried by a quadcopter and is used to localize the other quadcopters.

Processes and data flow. The rate of sending control commands to our autonomous toy-quadcopter must be of at least 30 FPS, even just to make sure that the quadcopters will not get crashed to the walls. The following pipe-line is computed independently for each of the quadcopters that are equipped with a camera.

(1) Capture the real-time video using the analog video receivers.

(2) Apply color filtering for (a) blue and purple, to detect the balloons; see Fig. 6a. (b) Only purple filtering, for marker detection.

(3) Apply Algorithm 1 to detect the blue circle (balloon) in the current frame.

(4) Estimate the purple circle (marker) on the detected balloon from the previous step. We used Open CV's blob detector [12].

(5) Predict the position of the neighbor quadcopters based on all detected circles, markers and the position of them, e.g. purple marker inside blue circle represents a neighbor quadcopter.

(6) Compute $6DOF$ of the observed quadcopter, based on the detected balloon and marker, as explained in Section 3.

(7) Compute the desired correction based on the computed $6DOF$. The correction are computed based

on relative positioning using its previous and current 6*DOF* via a standard PID controller [35].

(8) The correction commands are then sent to the remote controller via the Arduino board that is connected to the laptop as describe in the "Hardware" paragraph. The result is an autonomous hovering of each quadcopter, independently of the other quadcopters.

5. Experimental Results

We present experimental results on our system, for few example experiments as shown on our companion video [11].

We compared our results to existing implementations that were chosen by OpenCV for *HoughCircle* detection and *BlobDetection*. For each image frame, we calculated the *error* of each of the algorithms as follow; For an exact circle $C(x,y,r)$ and a detected circle $C^*(x^*,y^*,r^*)$, we define

$$error = \|x-x^*\| + \|y-y^*\| + \|r-r^*\|. \qquad (10)$$

Fig. 6a shows the detected circle during the experiment, which is marked by yellow. Fig. 6b shows a comparison of the calculated *error* between the algorithms over 500 frames. Fig. 6c shows an histogram of the corresponding *error*.

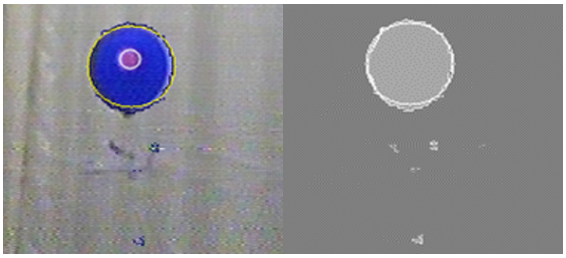
The above results show an average *error* of 1.73 pixels using our algorithm compared to 8.02 pixels of the *OpenCV* circle detection algorithms and 18.86 pixels of the *OpenCV* Blob detection. Moreover, the percentage of valid detection (i.e. $error < 5$) was 94.6% with our algorithm while *OpenCV* succeeds detection with less then 50%. Furthermore, we observed a trembling of 6.2 pixels with our algorithm, while *OpenCV* algorithms observed with more then 15 pixels of detection trembling

6. Conclusion

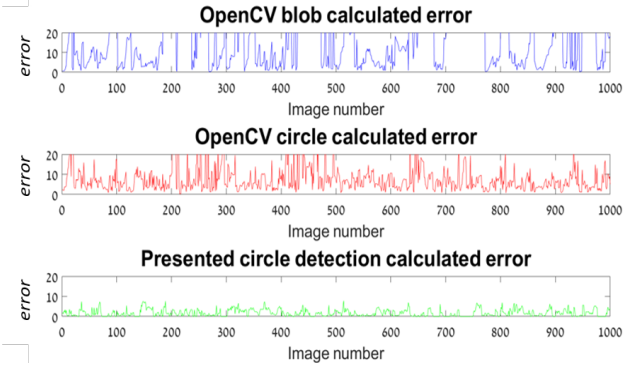
We suggested a real-time accurate and robust algorithm to detect convex shapes, and demonstrated it for detecting circle objects in digital images. The algorithm carefully selects a small subset of points (matchset) that approximates the desired shape, and track only those points to allow sub-linear run-time. This novel approach and selection significantly reduces the running time and therefore the process time for each frame. Moreover, we show how our algorithm is robust to outliers and to other image disorders. We then compared it to other state-of-the-art algorithms and showed that it is the only one that is sufficiently fast for our application, while still producing the desired approximation for our system.

Further work is to apply our algorithm for other shapes and other applications, as well as a swarm of many quadcopters outside the lab. We hope that our open code would encourage researchers to explore these research directions.

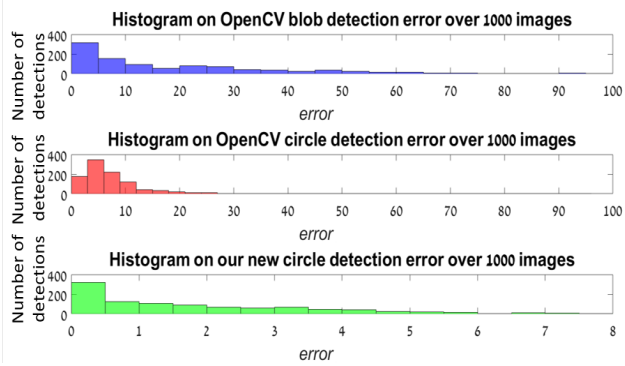
Figure 6: *Experimental results; (a) A snapshot of a circle detection on the captured image; (b) Comparison of calculated error; (c) Comparison of error histogram.*



(a)



(b)



(c)

References

[1] Andreas Koschan, Sangkyu Kang, Joonki Paik, Besma Abidi, and Mongi Abidi. Color active shape models for tracking non-rigid objects. *Pattern Recognition Letters*, 24(11):1751–1765, 2003.

[2] Thomas B Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer vision and image understanding*, 104(2):90–126, 2006.

[3] Vladimir J Lumelsky and Alexander A Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2(1):403–430, 1987.

[4] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[5] Dongsung Kim and Ramakant Nevatia. Recognition and localization of generic objects for indoor navigation using functionality. *Image and Vision Computing*, 16(11):729–743, 1998.

[6] Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. Visual navigation for mobile robots: A survey. *Journal of intelligent and robotic systems*, 53(3):263, 2008.

[7] Luciano da Fontoura Da Costa and Roberto Marcondes Cesar Jr. *Shape analysis and classification: theory and practice*. CRC Press, Inc., 2000.

[8] Zhiwei Zhu, Qiang Ji, Kikuo Fujimura, and Kuangchih Lee. Combining kalman filtering and mean shift for real time eye tracking under active ir illumination. In *Pattern Recognition, 2002. Proceedings. 16th International Conference On*, volume 4, pages 318–321. IEEE, 2002.

[9] Zhiwei Zhu, Kikuo Fujimura, and Qiang Ji. Real-time eye detection and tracking under various light conditions. In *Proceedings of the 2002 symposium on Eye tracking research & applications*, pages 139–144. ACM, 2002.

[10] Biswas and Veloso. Depth camera based indoor mobile robot localization and navigation. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1697–1702. IEEE, 2012.

[11] Haifa University Dror Epstein, RBD Lab. Experimantal movie of quadcopter tracks quadcopter, 2017. Available at https://youtu.be/-i2YIf2A0Ac.

[12] OpenCV Development Team. Opencv api reference. 2015. Available at http://docs.opencv.org/modules/core/doc/intro.html.

[13] Cuneyt Akinlar and Cihan Topal. Edcircles: A real-time circle detector with a false detection control. *Pattern Recognition*, 46(3):725–740, 2013.

[14] David A Forsyth and Jean Ponce. A modern approach. *Computer vision: a modern approach*, pages 88–101, 2003.

[15] C Rafael Gonzalez and Richard Woods. Digital image processing. *Pearson Education*, 2002.

[16] Christian Teutsch, Dirk Berndt, Erik Trostmann, and Michael Weber. Real-time detection of elliptic shapes for automated object recognition and object tracking. In *Electronic Imaging 2006*, pages 60700J–60700J. International Society for Optics and Photonics, 2006.

[17] Hough Paul VC. Method and means for recognizing complex patterns, 1962. US Patent 3,069,654.

[18] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[19] Erik Cuevas, Fernando Wario, Valentín Osuna-Enciso, Daniel Zaldivar, and Marco Pérez-Cisneros. Fast algorithm for multiple-circle detection on images using learning automata. *IET Image Processing*, 6(8):1124–1135, 2012.

[20] Xianen Zhou, Yaonan Wang, Qing Zhu, and Zhiqiang Miao. Circular object detection in polar coordinates for 2d lidar data. In *Chinese Conference on Pattern Recognition*, pages 65–78. Springer, 2016.

[21] Hanqing Zhang, Krister Wiklund, and Magnus Andersson. A fast and robust circle detection method using isosceles triangles sampling. *Pattern Recognition*, 54:218–228, 2016.

[22] Bing Zhou and Yang He. Fast circle detection using spatial decomposition of hough transform. *International Journal of Pattern Recognition and Artificial Intelligence*, 31(03):1755006, 2017.

[23] Joseph DeGol, Timothy Bretl, and Derek Hoiem. Chromatag: A colored marker and fast detection algorithm. *arXiv preprint arXiv:1708.02982*, 2017.

[24] Kuo-Liang Chung, Yong-Huai Huang, Shi-Ming Shen, Andrey S Krylov, Dmitry V Yurin, and Ekaterina V Semeikina. Efficient sampling strategy and refinement strategy for randomized circle detection. *Pattern recognition*, 45(1):252–263, 2012.

[25] Tommaso De Marco, Dario Cazzato, Marco Leo, and Cosimo Distante. Randomized circle detection with isophotes curvature analysis. *Pattern Recognition*, 48(2):411–421, 2015.

[26] Zhongyang Zheng, Bo Wang, Yakun Wang, Shuang Yang, Zhongqian Dong, Tianyang Yi, Cyrus Choi, Emily J Chang, and Edward Y Chang. Aristo: An augmented reality platform for immersion and interactivity. 2017.

[27] Filippo Bergamasco, Andrea Albarelli, Luca Cosmo, Emanuele Rodola, and Andrea Torsello. An accurate and robust artificial marker based on cyclic codes. *IEEE transactions on pattern analysis and machine intelligence*, 38(12):2359–2373, 2016.

[28] John Wang and Edwin Olson. Apriltag 2: Efficient and robust fiducial detection. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4193–4198. IEEE, 2016.

[29] Geometric idea of Parallax. Available at https://wikipedia.org/wiki/Parallax.

[30] Syma X5C Description. Available at http://symatoys.com/goodshow/x5c-syma-x5c-explorers.html.

[31] Walkera TX5805 Camera. Available at https://amazon.com/NEEWER-TX5805-Camera-Realtime-Display/dp/B00GUBQVEM.

[32] FAVOLCANO Easycap Converter. Available at https://amazon.com/FAVOLCANO-Easycap-Converter-Capture-Adapter/dp/B00LVTUX7E.

[33] Lenovo Laptop Description. Available at http://lenovo.com/psref/pdf/withdraw.

[34] Arduino Mini-computer Description. Available at https://arduino.cc/en/main/arduinoBoardUno.

[35] Amlan Basu, Sumit Mohanty, and Rohit Sharma. Introduction of fractional elements for improvising the performance of pid controller for heating furnace using amigo tuning technique. *Perspectives in Science*, 8:323–326, 2016.