2024 Tenth Indian Control Conference (ICC)
9 - 11 December, 2024
IISER Bhopal, India

Traversal of Unmanned Autonomous Vehicles Through Arbitrarily Shaped Openings

Karthickeyan V, Shashi Ranjan Kumar, and Hemendra Arya

Abstract— **In recent years, deploying unmanned aerial and ground vehicles has become increasingly prevalent across various domains, such as agriculture, defense, and filmmaking, owing to their versatility and efficiency. Despite their widespread use, these robots are still facing challenges in certain scenarios, particularly in navigating through arbitrarily shaped openings. This paper addresses the above challenge by proposing algorithms designed for opening detection and feasibility checks, ensuring safe passage through these complex environments. The core of this work revolves around the contour detection algorithm, developed using OpenCV, and the optimal rectangle extraction algorithm, which significantly enhances computational efficiency and is employed to fit an optimal rectangle within the detected contour. This step is crucial for ensuring the robot's safety by generating a feasible goal point for the point-tracking algorithm to follow. To validate the effectiveness of the proposed algorithms, a comprehensive experimental setup, including a ground robot equipped with various state estimation techniques, was established. Extensive testing and validation were conducted within this experimental framework, demonstrating the robustness and reliability of the algorithms in real-world scenarios. The successful implementation of these algorithms in hardware attests to this research's practical applicability and potential impact.**

Index Terms— **Gap detection, aerial and ground robotics, control algorithm, hardware implementation, perception and autonomy, visual tracking, and visual-based navigation.**

I. Introduction

The autonomous vehicles never fail to fascinate us, whether they are performing a simple, mundane task or embarking on a highly complex mission. The applications of these vehicles are widespread, and integrating them into every industry has also proven cost-effective and time-efficient. One such application is rescue operations during the aftermath of earthquakes or tsunamis, where autonomous vehicles must pass through arbitrarily shaped openings to collect necessary information and take required actions. In this paper, the primary objective is to address a seemingly simple yet crucial problem in unmanned aerial and ground vehicles (UAGV), that is, navigating a UAGV through an arbitrarily shaped opening situated in its path. This problem holds significance as it poses unique challenges related to UAGV navigation, especially when the shape of the opening is irregular and not predefined. The paper aims to develop and implement solutions using actual hardware, demonstrating a practical and real-world application of the proposed navigation methods.

Recent research on quadcopter gap traversal has brought fresh ideas to improve navigation in challenging environments. To enable quadrotors to navigate unfamiliar, angled gaps, [1] and [2] devised a deep reinforcement learning (DRL) approach employing onboard cameras. Xie et al. [1] achieved an 84.51% success rate for gaps up to 60 degrees without prior environmental knowledge. A simultaneous planning and execution (SPAE) technique was suggested by Liu et al. [3], which included online disturbance estimations for aggressive flight via slanted gaps. The technique was validated by outdoor trials. Penin et al. [4] designed dynamically viable trajectories that direct the drone to fly through the window based on the differential flatness property. By resolving posture uncertainty and active vision, Falanga et al., [5], coupled IMU data with single-camera gap detection, reaching an 80% success rate for gaps up to 45 degrees. Le et al. [6] achieved 85% accuracy up to 45 degrees by modifying trajectories in real-time during autonomous flying via small gaps using inertial and vision sensors. Robust posture estimation was introduced by Li et al. [7], while monocular vision-based navigation for UAVs in GPS-denied situations was introduced by Padhy et al. [8]. While Wang et al. [9] and Loianno et al. [10] concentrated on edge detection and lightweight quadrotor performance in demanding flight conditions, Sanket et al. [11] proposed a bioinspired perception framework, achieving an 85% success rate at 2.5 m/s using a monocular camera. Interestingly, rectangular gaps were assumed in most of these works, which is not necessarily true in practice. Algorithms for fitting the largest inscribed rectangle (LIR) within irregular shapes have been investigated to solve this problem. Sarkar [12] and Marzeh [13] suggested techniques for locating the LIR in digital objects and arbitrary polygons, respectively, while Datta [14] and Choi [15] created algorithms for isothetic polygons.

The research gap addressed in this project revolves around the need for developing a robust irregular opening detection system and formulating a control algorithm that allows the unmanned vehicle to navigate through such openings. A pivotal challenge lies in designing these algorithms to ensure efficient runtime execution, particularly when deployed on a resource-constrained platform like the Raspberry Pi. This study introduces a contour detection and optimal rectangle extraction algorithm to address the identified research gap. These algorithms are elaborated upon in Sections III-A and III-B, titled "Contour Detection Algorithm (CDA)" and

The authors are with the Department of Aerospace Engineering, Indian Institute of Technology Bombay, Powai, Mumbai 400076, India. (e-mails: 22m0020@iitb.ac.in, srk@aero.iitb.ac.in and arya@aero.iitb.ac.in).

This work was supported in part by the Ministry of Electronics and Information Technology, India, via Capacity building for Human Resource Development in Unmanned Aircraft Systems (Drone and related technology) - Guidance, Navigation, and Control Algorithms and Simulations (Reference No: L-14011/29/2021-HRD).

"Optimal Rectangle Extraction Algorithm (OREA)". The CDA aims to identify the arbitrarily shaped opening in the image frame and the OREA conducts the feasibility assessment for passage through the opening and performs safety checks. A proper experimental setup was established to validate the algorithms, including the construction of an unmanned ground vehicle (UGV) from scratch and the state estimation setup using an arUco tracker for increased reliability.

II. Problem Statement

Consider a scenario where an autonomous UAGV needs to traverse through an arbitrarily shaped opening as shown in the figure 1. To accomplish this, the algorithm should fulfill the following objectives:

i) The UAGV should be capable of detecting the edges of an arbitrarily shaped opening in a wall. The algorithm should then analyze the detected edges and form a closed contour to obtain the shape of the opening.
ii) After identifying the shape of the opening, the algorithm should incorporate a feasibility check to determine whether the UAGV can pass through the opening while maintaining an optimal safe distance from its edges.
iii) The UAGV should design its own trajectory and pass through the opening after extracting the opening's information.

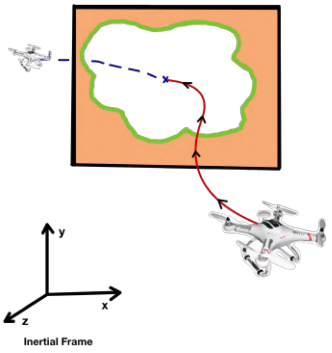


Fig. 1: Navigation of UAGV through irregular opening.

The aim of this paper is to enable the UAGV to traverse through the arbitrarily shaped opening without any human intervention. To achieve this goal, the UAGV must adhere to a certain set of procedures. These procedures are outlined in the figure 2 and must be executed in a specific sequence. The

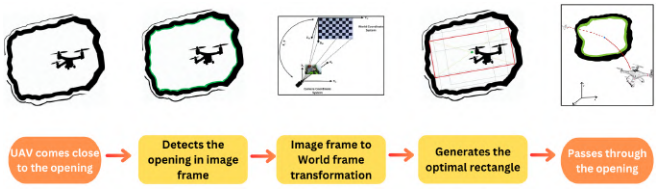


Fig. 2: Sequential execution of contour detection and traversal through the opening.

set of procedures that should be carried out by the UAGV is listed below:

- Begin by bringing the UAGV in close proximity to the opening
- Utilize an RGB-D camera and a contour detection algorithm to extract the contour of the opening
- Transform the contour coordinates obtained in the image frame to the world frame using the depth information obtained from the depth sensor
- Calculate an optimal rectangle within this transformed contour to enable the UAGV to navigate through it while maintaining maximum safety from the opening's boundaries. The optimal rectangle is the largest rectangle (in terms of area) within the contour, obtained with the given width and height constraints
- From the optimal rectangle, generate a goal point on the contour plane, which ensures the UAGV passes through the opening without any collision with the edges.
- Subsequently, the UAGV can accomplish its task by tracking the goal point under the guidance of a controller

III. Methodology

This section provides an in-depth analysis of the project workflow discussed in section II. The prime aspect covered in this work is the breakdown of the CDA and OREA, which have been built using the OpenCV package. This is followed by transformations and point tracking, which are crucial for hardware implementation.

A. Contour Detection

This algorithm is developed with the help of the OpenCV library to detect the opening contour seamlessly. Extracting the opening contour from the provided image based on the text may sound simple. It is not true, though. Getting the opening shape requires many modifications. Since the Realsense D455 camera is being used, both color images and depth images will be generated. Instead of using color images for contour detection, depth images can be used. This eliminates one of the main drawbacks of contour detection using color images, which necessitates the existence of thick contrasting borders along the opening edges. It is known that not all real-life openings have thick, contrasting borders. This slight change in the thought process brings the algorithm one step closer to real-life applications. The algorithm 1 is clearly explained in the following:

- Begin by converting the input image into a grayscale representation and then implement thresholding
- Enlarge the white pixels in the thresholded image through a dilation process, effectively filling gaps along the edges. This aids in simplifying contour formation
- Extract the closed contours from the dilated image using OpenCV's built-in `findContours()` function. However, not all contours are necessary in this context. Recognizing that the opening or gap contour is the outermost contour, only external contours are retained
- After extracting all external contours, another filtering phase becomes essential

- Fit all the selected contours with a bounding axis-aligned rectangle
- Calculate the extent, defined as the ratio of the area enclosed by the bounding rectangle to the contour's area
- Contours with an extent lesser than the constant 'C' are considered eligible, while others are not. After the second filtration, the count of eligible contours is likely reduced
- Identify the contour with the largest area among the eligible contours; this contour represents the opening contour

Algorithm 1: Contour Detection.

Input: Image from RGB-D sensor;
Output: Finalized contour coordinates;

```
Function ContourDetection:
    Processed Image = preprocessing(Input Image)
    Contours = findContours(Processed Image) for
     Contour within Contours do
        Rectangle = boundRectangle(Contour)
        Contour Area = area(Contour)
        Extent = area(Rectangle) / Contour Area
        if Extent <= C then
            Selected Contour = Contour
        Finalized Contour = max(area(Selected
         Contour))
        drawContour(Image, Finalized Contour)
    return Image, Finalized Contour
```

B. Optimal Rectangle Extraction

Once the contour is transformed to the world frame using the transformations, the next step is to find whether the opening can accommodate the UAGV. In other words, a feasibility check has to be done to check whether it is possible for the robot to pass through the opening without collision. The next challenge is to devise a way to conduct this feasibility test and get the goal point coordinates in the inertial frame.

Consider the robot as a cuboid (which is a valid assumption) with known dimensions. It is evident that the cross-section of the robot is a rectangle. The problem is simplified to ensure that a robot's rectangular cross-section can traverse through the opening without any collision. The known parameters include the coordinates of the contour in the image frame and the dimensions of the cuboid (that is, robot).

The OREA is designed to obtain the optimal rectangle for the given opening and UAGV information. This algorithm 2 is explained as follows:

- Obtain the smallest rectangle containing the given contour
- Divide the rectangle into small rectangles using the contour vertices. This step is depicted in the figure 3a which gives a clear picture
- Iterate through all the small rectangles and find out the location of each rectangle (inside or outside the contour). This can be done using the ray tracing algorithm. Once it is done, the figure 3a will be transformed to figure 3b
- Obtain the horizontal and vertical adjacency vectors for each rectangle within the contour. The horizontal adjacency vector represents the distance in terms of small rectangles that can be traversed by the selected rectangle horizontally with respect to each row. The vertical adjacency vector is the same as horizontal adjacency, but this represents the vertical traversal distance in terms of small rectangles. The figure 3c explains the same step in a pictorial way
- Using the adjacency information, calculate the largest possible rectangles for each root cell as shown in the figure 3d and append it to a list. Now, the list contains all the possible rectangles within the given contour
- Iterate through all the rectangles and find the feasible rectangles (if any) using the known information of the approximated cuboid (robot)
- Obtain the optimal rectangle (if it exists) whose width and height should be greater than the cross-section of the approximated cuboid and whose area should be the maximum among all the feasible rectangles

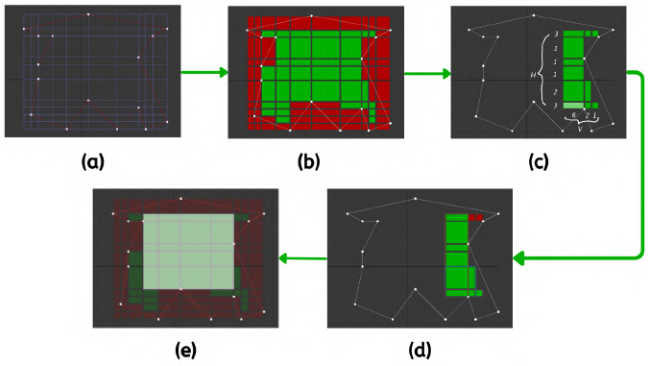


Fig. 3: OREA flow chart [16].

When the algorithm is provided with polygon coordinates as input, the runtime exceeds 160 ms. It is more efficient to use a binary grid as input. However, the auto-generated binary grid produced by the provided algorithm [17] is considerably large in size, resulting in increased runtime. A binary grid generation function has been developed to address this issue, which provides a smaller, approximated grid. As a result, the runtime is significantly reduced to just 2 ms, making the algorithm 80 times faster and suitable for implementation on Raspberry Pi.

C. Point Tracking

Once the optimal rectangle is generated for the contour in the image frame, the center point of this rectangle is taken as the goal point. This goal point is then given to the point tracking algorithm to enable the UAGV to reach the destination, that is, pass through the opening safely. Instead

Algorithm 2: Optimal Rectangle Extraction.

Input: Contour; Area, width, and height of the desired rectangle;
Output: Coordinates of feasible rectangles and optimal rectangle;

```
Function GridGeneration:
    for rectangle in small rectangles do
        for point in rectangle do
            point_bool = IsInside(point)
        if all_point_bool is True then
            interior_rectangles.append(rectangle)
        else
            Continue
    return interior_rectangles
Function OptimalRectangle:
    grid = GridGeneration(contour)
    possible_rectangles = LIR(grid)
    for rectangle in possible_rectangles do
        if (area > desired area) and (width > desired width) and (height > desired height) then
            feasible_rectangles.append(rectangle)
            if area > max_area then
                optimal_rectangle = rectangle
                max_area = area
    return optimal_rectangle
```

Algorithm 3: Point Tracking.

Input: Goal point;
Output: Ground robot reaching the goal point;

```
while True do
    get the ground robot states (x, y, θ) if there is no angular and linear error between ground robot and goal point then
        Reached goal...
    if there exists an angular error between the ground robot and goal point then
        give angular velocity command to rotate the robot to align with the goal point
    else
        give linear velocity command to move the robot to reach the goal point
```

of relying on data from a single frame, multiple frames were used to generate 30 goal points. All 30 goal points should approximately point toward the same destination and should be verified by comparing these goal points with each other. The explanation of the algorithm 3 is given below:

- Receive input goal coordinates
- Begin continuous state estimation
- Use trigonometry to calculate the required heading angle for the UAGV to reach the goal in a straight line. Also, calculate the distance to be traveled using the distance formula
- Orient the UAGV along the required heading angle and start moving toward the goal
- In case the UAGV deviates from the straight path due to any reason and the difference between the current heading and the required heading is more than 8 degrees (adjustable), the UAGV should do a counter maneuver
- The UAGV should continue moving forward until the required distance is traversed and the goal is reached
- Stop once the goal has been reached

IV. Results and Discussion

This section presents the experimental setup and hardware demonstrations to validate the algorithms proposed in Section III, using actual hardware, and the corresponding inferences are discussed.

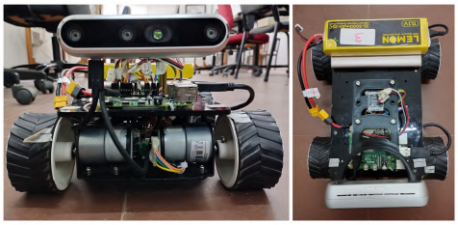


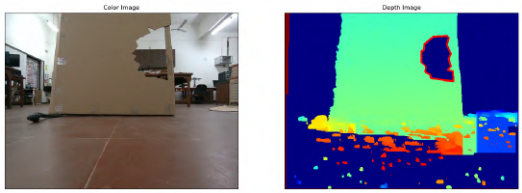
Fig. 4: Front (left) and top (right) view of the rover.

A. Experimental Setup

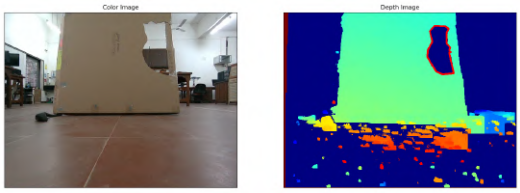
The algorithms proposed in the Section III are designed for UAVs, but for the validation of the algorithm, it is safer to implement in the ground robot, which minimizes the cost and time caused in terms of crashes. Additionally, validating the algorithm using the ground robot implies that the algorithm also works for UAVs with slight modifications. The figure 4 depicts the rover with all components integrated, that has been built from scratch. The GB37-520 12V DC motors are used, and the speed and direction control are managed with the help of L298N-based 2-channel motor driver modules. The Raspberry Pi 4 Model B acts as the brain of the rover, where all computations take place. The opening is detected using the Intel RealSense D455 camera, which provides the RGB and depth images. The arUco tracker is deployed to provide the instantaneous states of the rover with sufficient accuracy. The overall dimensions of the rover are 30×15 cm (with the arUco marker hat). Arbitrarily shaped openings are carved in large cardboards, and the size of the openings is approximately 40 cm in width and 25 cm in height. A width and height tolerance of 10 cm is given to accommodate the image-to-world-frame transformation and state estimation errors.

B. Contour Detection Algorithm

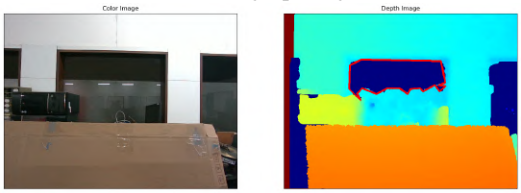
The results obtained using the Contour detection algorithm (section III-A) are presented in the figures 5a to 5c. In these figures, the red line represents the opening's contour, and it



(a) Floating opening 1.



(b) Floating opening 2.



(c) Floating opening 3.

Fig. 5: Performance using contour detection algorithm.



Fig. 6: Optimal rectangle detection - Testcases 1 and 2.

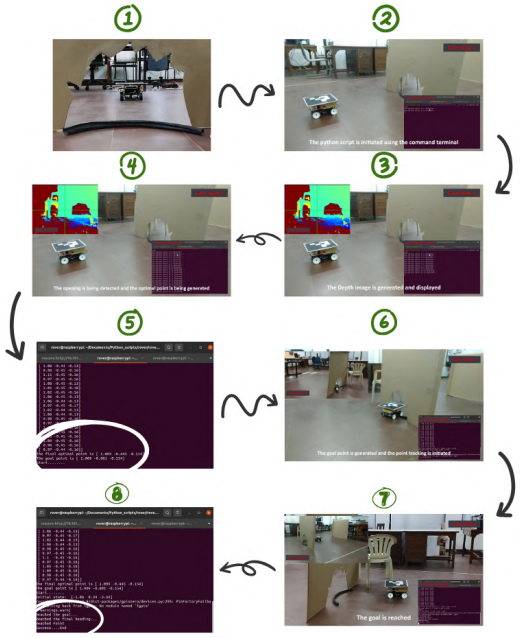
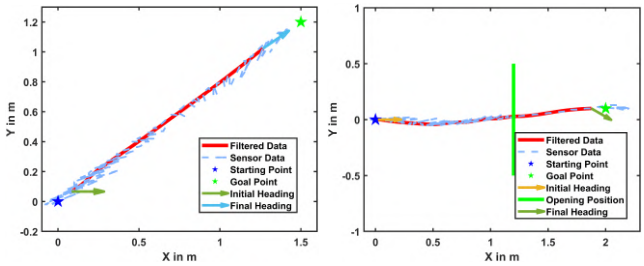


Fig. 7: Holistic implementation using robots.



(a) Point tracking. (b) Holistic implementation.

Fig. 8: The path traced by the ground robot.

is evident that the algorithm detects the contour of both the experimental test cases (figures 5a and 5b) and real openings (figure 5c). From the figures in figures 5a and 5b, it can be inferred that the opening is being detected even without thick and dark contrasting outer borders around it. The test cases in figures 5a to 5c are named floating openings because the opening appears to be hovering in the air. These types of openings are more suitable for a quadcopter to pass through, and the algorithm easily detected the floating openings. Also, note that the figure 5c ticks the reality checkbox where the opening is actually a ventilation window large enough for UAGV to pass through, and the algorithm detects the opening from the depth image easily.

C. Optimal Rectangle Extraction

The optimal rectangle extraction algorithm (2) has been implemented for various test cases, and the results are available in figure 6. In these figures, the green line represents the opening's contour, the blue rectangles represent feasible rectangles, and the red rectangle represents the optimal rectangle.

D. Point Tracking Algorithm

The point-tracking algorithm enables the ground robot to reach the goal point. Initially, the ground robot aligns itself towards the opening, and then travels straight until the goal is reached. The corresponding plot is depicted in the figure 8a. The error tolerance is chosen to be 10 centimeters for linear distance error and 8 degrees for angular distance error. When the ground robot reaches a radius of 10 cm from the goal with an angular error of 8 degrees, then the goal is considered to be achieved.

The states are obtained from the arUco tracker, and as expected, the sensor data is noisy. Using the Gaussian filter in MATLAB, the data is smoothed and plotted as shown in figure 8a. The starting and goal points are represented by the blue and green stars, respectively, and the initial and final headings are denoted with the green and light blue arrowheads. It can be seen from figure 8a that the ground robot successfully reaches the goal point, and the states are within the error tolerance values.

E. Holistic Implementation

The holistic demonstration is the sequential compilation of all the algorithms explored in section III. This implementation is explained in the following with the help of the figure 7. The rover is positioned such that the opening is within the field-of-view of the RGB-D sensor, as shown in figure 7 - 1. All the algorithms are sequentially compiled into a single Python file called `main.py`. After initiating this file, the ground robot becomes completely autonomous and completes its task. Once the main file is initiated (figure 7 - 2), the color and depth images (figure 7 - 3) are generated by the Realsense RGB-D camera. The images are fed into the contour detection algorithm (section III-A), and the contour coordinates are collected in the image frame as an outcome (figure 7 - 4). The contour coordinates in the world frame are procured by plugging the pixel coordinates of the contour into the transformations. The optimal rectangle is fitted inside the contour coordinates in the world frame, and the goal point is generated (figure 7 - 5) using the algorithm 2. The point tracking algorithm (section III-C) is initiated as seen in figure 7 - 6, enabling the ground robot to pass through the opening without any collisions. The traces of the ground robot in the world frame is depicted in figure 8b. Finally, the ground robot traverses through the opening and stops after crossing it (figure 7 - 7 and 8).

The video demonstrations are performed with various initial conditions and different arbitrary openings, and the corresponding video links are listed below:

- Arbitrary opening 1 with point tracking plot.
- Arbitrary opening 2 with point tracking plot.
- Arbitrary opening 1 with the ground robot positioned at the leftmost side of the opening.
- Arbitrary opening 2 with the ground robot positioned at the rightmost side of the opening.

V. Conclusion

In this paper, the UAGV successfully navigated through openings without prior knowledge of their shape, relying on a controller that can be implemented in actual hardware. The complete objective of the problem stated earlier was implemented in the ground robot. From contour detection to passing through the opening, all steps were clearly explained, and the results for each algorithm were showcased and validated in section IV. Since the contour selection algorithm works smoothly for the ground robot implementation, it can be inferred that it will also work for a quadcopter. In summary, the current progress involves the successful completion of contour detection for the opening, transforming the contour to the world frame, precisely inscribing an optimal rectangle within the contour in the world frame to ensure a maximum safe distance from the opening's edges, generating the goal point, and tracking the goal point with the 4WD ground robot. One of the assumptions made during the development of the algorithm is that the opening is within the field-of-view (FOV). A precursor to this work could involve determining the likelihood of an opening or window in the deployed area. If the likelihood is high, the quadcopter could adjust to bring the opening within its FOV to implement the aforementioned algorithms. This approach can bridge the gap between the experimental setup and real-life scenarios, such as detecting broken windows.

Acknowledgement

We thank Mr. Jash Devang Desai for the insightful discussion and help in performing experiments.

References

[1] Y. Xie, M. Lu, R. Peng, and P. Lu, "Learning agile flights through narrow gaps with varying angles using onboard sensing," *arXiv preprint arXiv:2302.11233*, 2023.

[2] J. Lin, L. Wang, F. Gao, S. Shen, and F. Zhang, "Flying through a narrow gap using neural network: an end-to-end planning and control approach," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 3526–3533.

[3] Z. Liu and L. Cai, "Simultaneous planning and execution for quadrotors flying through a narrow gap under disturbance," *IEEE Transactions on Control Systems Technology*, vol. 31, no. 6, pp. 2644–2659, 2023.

[4] B. Penin, P. R. Giordano, and F. Chaumette, "Vision-based reactive planning for aggressive target tracking while avoiding collisions and occlusions," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3725–3732, 2018.

[5] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision," in *International Conference on Robotics and Automation*. IEEE, 2017, pp. 5774–5781.

[6] N.-H.-D. Le, H.-P. Le, T.-N.-M. Ta, T.-D. Trinh, and M.-H. Le, "Narrow gaps detection and computing trajectory for aggressive autonomous objective-navigating quadcopter," in *International Conference on Green Technology and Sustainable Development*, 2022, pp. 454–458.

[7] S. Li, M. M. Ozo, C. De Wagter, and G. C. de Croon, "Autonomous drone race: A computationally efficient vision-based navigation and control strategy," *Robotics and Autonomous Systems*, vol. 133, p. 103621, 2020.

[8] R. P. Padhy, F. Xia, S. K. Choudhury, P. K. Sa, and S. Bakshi, "Monocular vision aided autonomous UAV navigation in indoor corridor environments," *IEEE Transactions on Sustainable Computing*, vol. 4, no. 1, pp. 96–108, 2018.

[9] X. Wang, Y. Deng, and H. Duan, "Edge-based target detection for unmanned aerial vehicles using competitive bird swarm algorithm," *Aerospace Science and Technology*, vol. 78, pp. 708–720, 2018.

[10] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, "Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and IMU," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 404–411, 2016.

[11] N. J. Sanket, C. D. Singh, K. Ganguly, C. Fermüller, and Y. Aloimonos, "Gapflyt: Active vision based minimalist structure-less gap detection for quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2799–2806, 2018.

[12] A. Sarkar, A. Biswas, M. Dutt, and A. Bhattacharya, "Finding a largest rectangle inside a digital object and rectangularization," *Journal of Computer and System Sciences*, vol. 95, pp. 204–217, 2018.

[13] Z. Marzeh, M. Tahmasbi, and N. Mirehi, "Algorithm for finding the largest inscribed rectangle in polygon," *Journal of Algorithms and Computation*, vol. 51, no. 1, pp. 29–41, 2019.

[14] A. Datta, "Efficient algorithms for the largest rectangle problem," *Information sciences*, vol. 64, no. 1-2, pp. 121–141, 1992.

[15] Y. Choi, S. Lee, and H.-K. Ahn, "Maximum-area and maximum-perimeter rectangles in polygons," *Computational Geometry*, vol. 94, p. 101710, 2021.

[16] Evryway, "Finding the axis aligned largest interior rectangle in a simple polygon," The Blogpost - Largest interior rectangle, cited on 10th June 2024.

[17] L. A. Weber, "Largest interior rectangle," https://github.com/OpenStitching/lir/tree/main, cited on 10th June 2024.