AIP Conference Proceedings

Optimized autonomous UAV design with obstacle avoidance capability

Cite as: AIP Conference Proceedings **2233**, 020026 (2020); https://doi.org/10.1063/5.0001372
Published Online: 05 May 2020

Jun Jet Tai, Swee King Phang, and Yen Myan Felicia Wong

View Online Export Citation

Lock-in Amplifiers up to 600 MHz
Watch


AIP Publishing

© 2020 Author(s).

Optimized Autonomous UAV Design with Obstacle Avoidance Capability

Jun Jet Tai [1], Swee King Phang [1, a)] and Yen Myan Felicia Wong [1]

[1] *Taylor's University, 1, Jalan Taylors, 47500 Subang Jaya, Selangor, Malaysia*

[a)] *Corresponding Author: sweeking.phang@taylors.edu.my*

Abstract. Obstacle Avoidance and Navigation (OAN) algorithms are an active research field dominated by either offline or online methods. The former method is fast but requires a prior known map while the latter method can function without a prior known map at the expense of high computational requirements. To bring OAN algorithm to mass produced mobile robots, more precisely multirotor Unmanned Aerial Vehicles (UAVs), the computational requirement of robust algorithms must be brought low enough such that the computation can be done on an onboard companion computer, while being able to operate without prior knowledge of the map. This article introduces a novel OAN algorithm – dubbed the Closest Obstacle Avoidance and A* Algorithm (COAA*) – that bridges the capabilities of current offline and online OAN algorithms. The proposed algorithm takes into account the UAV performance limits, and is very easy to calibrate and incorporate for many other classes of mobile robots. The main contributions of this research work are that COAA* has guaranteed convergence to a global minimum for the navigational trajectory, while being very computationally lightweight due to its first principles formulation.

INTRODUCTION

The race to enable completely autonomous obstacle avoidance and navigation on unmanned aerial vehicles (UAVs) is one that is participated by many parties. Commercial UAVs operate in closed environments largely through a human operator. With the coming implication of many enabling technologies, the advent of UAV operation for the assistance of daily life is becoming an integral part of our society. Some major enabling technologies include advanced real-time connectivity through 5G networking [1], vision processing technologies through Artificial Intelligence [2], scalable information backbones through blockchain-based Internet-of-Things (IoT), and advanced UAV control schemes [3]. However, with increasing numbers of UAVs in operation, the need for intelligent autonomy is ever increasing [4]. This is especially true for UAVs intended to be operated within closed urban landscapes – Kuala Lumpur or New York city, indoor environments – a factory floor or an office building, or even search and rescue environments such as forested regions or post-disaster zones. In these instances, a key component of UAV systems is for them to operate autonomously without human assistance. Under this operation regime, many tasks such as UAV-based delivery, automated search and rescue, and factory floor inventory keeping, can be alleviated from the need of human labor, freeing up human resources for more crucial tasks.

Current OAN algorithms can be loosely classified into offline, online, and motion planning methods. Offline methods are generally based on the fundamental path finding algorithm of A*. These algorithms work by first assuming that knowledge of a global map is available, then encoding each empty space in the map as a series of nodes. A path from the beginning to a desired target point is then formulated, and multitudes of smoothing functions layered over the path to generate a flyable trajectory for the UAV. Computing the actual path itself is usually a computationally lightweight task. However, smoothing functions tend to be very computationally intensive to be computed, and hence done on an offboard workstation PC [5]. This requirement disallows running such algorithms on a resource-constrained platform such as a UAV.

13th International Engineering Research Conference (13th EURECA 2019)
AIP Conf. Proc. 2233, 020026-1–020026-12; https://doi.org/10.1063/5.0001372
Published by AIP Publishing. 978-0-7354-1992-6/$30.00

Attributed to the low computational requirement of running an algorithm online, online algorithms are largely simple in nature, utilizing simple rules to achieve obstacle avoidance while simultaneously heading towards a target. Of the many online methods, the simplest and most prominent forms are usually derivations of the famous potential field algorithm [6]. The variations are plenty, with examples being employed in unicycle style robots [7], high speed vehicles [8], as well as UAVs [9]. However, in order for guaranteed convergence of the robot to the target, the notion that the world is known and well described is a necessity. Potential field algorithms are also not well suited to UAV applications. The inability for producing curving paths around objects is an inherent property of many quickly computable potential fields. This inadvertently will force the UAV into slowing down when encountering an obstacle, as opposed to simply redirecting its velocity vector [10]. Another disadvantage of potential field algorithms are that the possibility of trajectories leading to a local minimum, away from the target point, is possible [11]. Vector fields as a whole can be described as a direct response to the downsides of potential field algorithms [12]. These algorithms are able to elegantly formulate a UAV trajectory friendly path by accounting for the conservation of velocity vectors [13]. Conversely, they still experience the same pitfalls that bewitch potential field algorithms – lack of guaranteed convergence to a global trajectory minimum. In a non-trivial manner, guaranteed convergence is only possible with the notion of a global map being known.

The challenge of a robust OAN algorithm has also been tackled by non-first principle derived techniques. These techniques generally do not need explicit mathematical modelling to describe and produce meaningful trajectories. The simplest of such algorithms utilize fuzzy logic to generate intuitive rulesets in pursuit of a target. When encountering unforeseen obstacles, the robot analyzes the navigational task in terms of primitive behaviors to determine the next best move [14]. The similar idea can be extended to multi-agent systems to allow greater observability, and also better decision making towards execution of a global objective [15]. Unfortunately, due to the rule-based approach of fuzzy logic, the presence of local minimum is still not guaranteed to be eliminated. The same is true for other modern OAN algorithms such as those which leverage the use of neural networks [14-15]. Furthermore, one large disadvantage holds back the implementation of these algorithms – that of training and implementation, which prevents these algorithms to be easily deployable on a large variety of robotics systems with different sensor-actuator suites. The gap for robust OAN algorithms on UAV platforms is therefore evident, and aptly introduced here are the contributions of the novel OAN algorithm presented in this work – dubbed COAA*:

1. Guaranteed convergence to a global minimum without any prior knowledge of a global map.
2. Easy implementation without needing any prior training for the robotics platform used.
3. Scalable to various sensor-actuator formats for various robotics system.
4. Low computational effort, able to run at a minimum of 20 Hz onboard a UAV.
5. Able to take into account UAV dynamics and generate smooth trajectories on the fly.

The remainder of this paper is partitioned into three sections. The groundwork for COAA* was first laid out in the form of the overall architecture. This is followed by an overview of the implementation and concept validation phase. The next section highlights some of the more technical results of the algorithm in action, and the article finalizes with conclusions and final thoughts.

COAA* ALGORITHM ARCHITECTURE

The principle assumption for algorithm development is that the map is perceived through an occupancy grid of the form $M_{\mathrm{L}}(x, y)$, which is updated based on perception of the global map $M_{\mathrm{G}}(x, y)$ as the UAV experiences over its time of operation. The rate of obstacle perception is dependent on an observability radius centered around the UAV, with a radius magnitude in the form of R_{OBS}. Thus, as the UAV traverses more and more of the map, $M_{\mathrm{L}} \rightarrow M_{\mathrm{G}}$ as $t \rightarrow \infty$. For sake of literacy, an occupied element in the occupancy grid is referred to as 'obstacle'.

COAA* is a first principles derived algorithm that takes inspiration from many existing OAN algorithms. The strategy is to utilize a hybrid architecture comprising of a localized vector field and potential field general purpose traverser, assisted in navigation by a non-linear path finding algorithm. This will not guarantee that the shortest path to the target is taken (such a feat is impossible without knowing the whole map), but will guarantee convergence to the target while considering UAV flight dynamics.

This characteristic is achieved by leveraging the velocity vector conserving traits of vector field techniques, while layering a velocity magnitude contributor in the form of a potential field. A first order velocity filter is then placed over the whole setup to decouple UAV dynamics constraining from the obstacle avoidance implementation. A path finding algorithm – in the form of a modified A* algorithm – is then built over the whole algorithm by allowing it to construct flight waypoints. This A* algorithm takes map information from a local map database onboard the UAV, that is constantly updated as the UAV traverses the course and learns about the map. Figure 1 is the overall algorithm flow architecture.

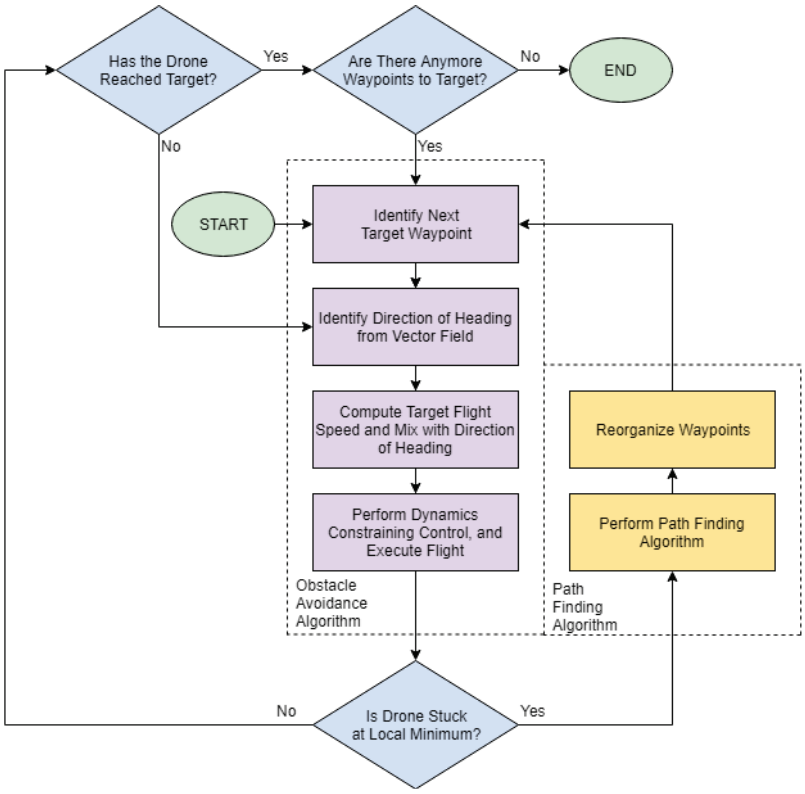


FIGURE 1. COAA* algorithm architecture.

Waypoint Generation

Graph search techniques are intrinsic to path finding challenges. In this paper, a modified A* algorithm is used to develop the connected path from the start to the target within the flight space. Additional modifications to the algorithm include adding a herein called 'danger' cost, which denotes the proximity of the connected paths to the obstacles within the flight space. The danger cost is important as it allows the UAV to prioritize wide open spaces where it is able to fly at faster speeds, effectively promoting safety and leveraging the efficiencies of faster velocity flight. Since no prior map information is given to the algorithm, the A* implementation only utilizes the information that is known to the UAV, and is re-executed every time the algorithm deems that progress has halted and hence the UAV is stuck at a local minimum. This method allows the A* algorithm to constantly be run with more and more available map information each run, as the UAV traverses and learns more of the map. The result of the connected path is the put through a split and merge (SaM) algorithm. Fundamentally, a SaM algorithm is one that iteratively segments a connected path by determining if the summed error of each segment exceeds a set threshold. A visual illustration of a SaM algorithm is shown in Figure 2. Formally, the summed error, E is denoted as

$$E = \frac{1}{P_{i+1} - P_i} \sum_{n=P_i}^{P_{i+1}} (Y_n - m_i X_n + C_i)^2 \tag{1}$$

P_i denotes the index of the i-th node from the waypoints list on the path list, Y_n is the y-value of the n-th point on the connected path, X_n is the corresponding x-value, and m_i and C_i are the gradient and intercept values for the i-th node on the waypoints list on the path list. The purpose of segregation a series of waypoints is twofold. The first is to allow the UAV to fly more natural paths intrinsic to UAV dynamics. In this manner, the SaM serves to decouple the UAV flight dynamics from the tightly defined connected path generated by the A* search algorithm (evidenced in Figure 3 to contain many right-angled turns). The second purpose of the SaM is to allow leeway for undiscovered obstacles to show up in the connected path itself. As a consequent to this, A* is not required when simple solutions to obstacle avoidance exists, and only called when the UAV is truly stuck. The outcome of the waypoint generation implementation is shown in Figure 3.

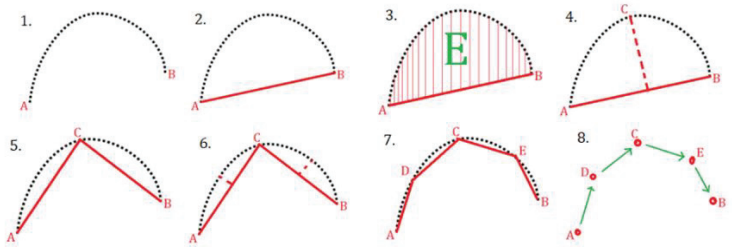


FIGURE 2. Split and merge algorithm.

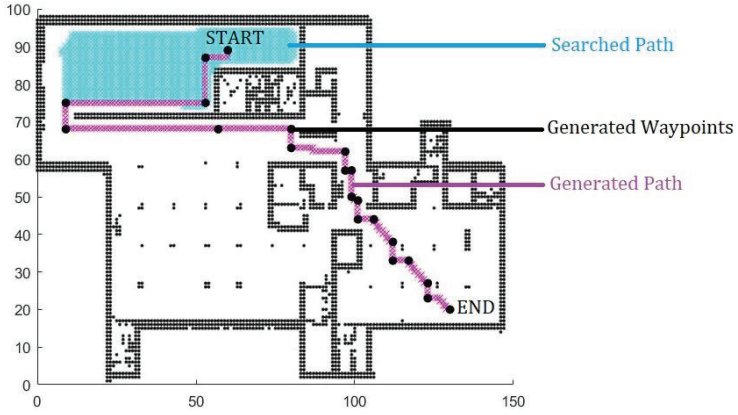


FIGURE 3. Waypoint generation implementation.

Obstacle Avoidance

In the case of unforeseen obstacles appearing within the generated path during A*, a very low computational cost obstacle avoidance implementation is used. This obstacle avoidance state is designed to be the main operating regime of the whole system. The algorithm is designed to look at the two closest obstacles within the occupancy grid, denoted here as $O_i \in \{O_1, O_2\}$. The desired flight velocity is then computed through a series of vector computations:

$$\boldsymbol{u}_{\mathbf{O2T}_i} = \frac{\boldsymbol{P}_\mathbf{T} - \boldsymbol{P}_{\mathbf{O}_i}}{|\boldsymbol{P}_\mathbf{T} - \boldsymbol{P}_{\mathbf{O}_i}|} \tag{2}$$

$$\boldsymbol{u}_{\mathbf{P2O}_i} = \frac{\boldsymbol{P}_{\mathbf{O}_i} - \boldsymbol{P}_\mathbf{P}}{|\boldsymbol{P}_{\mathbf{O}_i} - \boldsymbol{P}_\mathbf{P}|} \tag{3}$$

$$\boldsymbol{u}_{\mathbf{T2P}} = \frac{\boldsymbol{P}_\mathbf{T} - \boldsymbol{P}_\mathbf{P}}{|\boldsymbol{P}_\mathbf{T} - \boldsymbol{P}_\mathbf{P}|} \tag{4}$$

All vectoral notations are in $\mathbb{R}^2$. $\boldsymbol{P}_{\mathbf{O}_i}$represents the position of the i-th closest obstacle, where $i \in \{1,2\}$ representing the closest and second closest obstacle, $\boldsymbol{P}_\mathbf{P}$ and $\boldsymbol{P}_\mathbf{T}$ denote the position of the UAV and target respectively.

Consequently, $\boldsymbol{u}_{\mathbf{T2P}}$ denotes the unit vector from target to UAV position, and $\boldsymbol{u}_{\mathbf{P2O}_i}$ and $\boldsymbol{u}_{\mathbf{O2T}_i}$ represent the unit vector from UAV position to obstacle, and obstacle to target respectively. These vectors are used to compute:

$$\theta_i = atan2\left(\boldsymbol{u}_{\mathbf{O2T}_{i,x}}, \boldsymbol{u}_{\mathbf{O2T}_{i,y}}\right) - atan2\left(\boldsymbol{u}_{\mathbf{P2O}_{i,x}}, \boldsymbol{u}_{\mathbf{P2O}_{i,y}}\right) \tag{5}$$

$$\boldsymbol{u}_{\mathbf{UAV}_i} = \begin{cases} \begin{bmatrix} \boldsymbol{u}_{\mathbf{P2O}_{i,y}} \\ -\boldsymbol{u}_{\mathbf{P2O}_{i,x}} \end{bmatrix} & 0 < \theta < \frac{\pi}{2} \\ \begin{bmatrix} -\boldsymbol{u}_{\mathbf{P2O}_{i,y}} \\ \boldsymbol{u}_{\mathbf{P2O}_{i,x}} \end{bmatrix} & -\frac{\pi}{2} < \theta < 0 \\ \begin{bmatrix} \boldsymbol{u}_{\mathbf{T2P}_x} \\ \boldsymbol{u}_{\mathbf{T2P}_y} \end{bmatrix} & \text{otherwise} \end{cases} \tag{6}$$

Where $\boldsymbol{u}_{\mathbf{UAV}_i}$ corresponds to the computed UAV heading based on the i-th closest obstacle. Formally, Equations (2) to (6) describe the generation of the target heading generated on a per-obstacle basis. The visual representation of the vector field generated around an obstacle due to Equations (2) to (6) is shown in Figure 4(a). Ultimately, the computed headings based on i-th onstacles will be fused through:

$$\boldsymbol{u}_{\mathbf{UAV_C}} = \frac{\left(\boldsymbol{u}_{\mathbf{UAV}_1} \cdot D_2 + \boldsymbol{u}_{\mathbf{UAV}_2} \cdot D_1\right)}{\left|\left(\boldsymbol{u}_{\mathbf{UAV}_1} \cdot D_2 + \boldsymbol{u}_{\mathbf{UAV}_2} \cdot D_1\right)\right|} \tag{7}$$

$\boldsymbol{u}_{\mathbf{UAV_C}}$ represents the desired UAV heading based on the precomputed individual headings, and D_i represents the distance from the i-th obstacle to the UAV. Equation (7) describes the normalizing of desired headings generated from the two nearest obstacles, this is done to eliminate limit cycles from potentially occurring in the system. To prevent the UAV from colliding with obstacles that are closer than defined by D_{prox}, a push back vector is introduced via:

$$\lambda = \begin{cases} \frac{D_{\text{prox}}}{D} & D < D_{\text{prox}} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

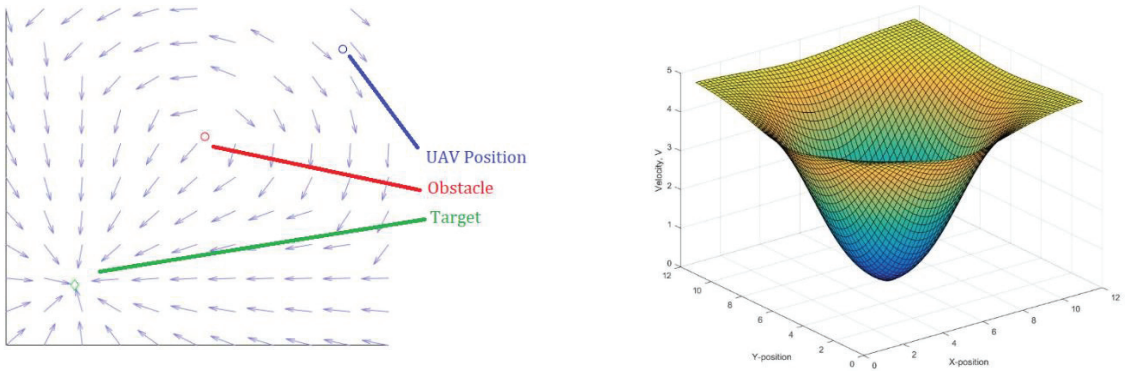
$$\boldsymbol{u}_{\mathbf{UAV_T}} = \frac{\lambda \cdot \boldsymbol{u}_{\mathbf{-H2T}} + \boldsymbol{u}_{\mathbf{UAV_C}}}{\left|\lambda \cdot \boldsymbol{u}_{\mathbf{-H2T}} + \boldsymbol{u}_{\mathbf{UAV_C}}\right|} \tag{9}$$

λ is a scaling factor for $\boldsymbol{u}_{\mathbf{-H2T}}$ which only acts when the UAV comes too close to an obstacle. Equation (9) describes the mixing method used to incorporate the push back vector into the desired UAV heading vector, and this produces the targeted UAV heading, $\boldsymbol{u}_{\mathbf{UAV_T}}$. Lastly, the desired UAV velocity vector, $\boldsymbol{V}_{\mathbf{UAV_T}}$, representing both velocity magnitude and direction, is computed via:

$$V_{\text{a}} = V_{\max} \cdot \frac{1}{1 + e^{k\left(-D_1 + D_{\text{prox}}\right)}} \tag{10}$$

$$\boldsymbol{V}_{\mathbf{UAV_T}} = \boldsymbol{u}_{\mathbf{UAV_T}} \cdot V_{\text{a}} \tag{11}$$

Where V_{a} represents the allowable flight velocity, computed based on the sigmoid function which depends on a steepness gradient, k. D_{prox} denotes the allowable proximity with which the UAV is allowed to approach an obstacle. Equation (10) describes the potential field around the nearest obstacle used to compute the allowable flight velocity of the UAV, with the potential field shown in Figure 4(b). Equation (11) is tying together all the computations to form the final velocity vector. The implementation used in this research is novel compared to many other similar works. The primary advantage being that the computation is fundamentally explicit, thus freeing the algorithm from the constraints of numerical solvers. In this manner, the main operating regime of the algorithm is free of numerical instabilities of which optimization algorithms suffer from. This also eliminates the need for extra checking and error handling during generation of the dynamic UAV trajectory, freeing up large amounts of computational power for more resource intensive tasks such as Simultaneous Localization and Mapping (SLAM).



(a) (b)

FIGURE 4. (a) Vector field around an obstacle as a representation of velocity heading. (b) Potential field around an obstacle centered at [5, 5] as a representation of allowable flight velocity. The closer the position of the UAV to the obstacle, the lower the allowable flight velocity. Via color, blue represents 0 allowable velocity, and yellow represents maximum velocity.

Dynamics Constraining

To further compensate for the potentially discontinuous desired velocity profiles, the velocity command given to the UAV is computed through a first order low pass filter, this technique smooths out the desired trajectory by minimizing control errors. The first order filter is represented as the following Laplace domain equation:

$$\frac{\boldsymbol{V}_{\mathbf{UAV_O}}}{\boldsymbol{V}_{\mathbf{UAV_T}}} = \frac{1}{1 + Ks} \tag{12}$$

$\boldsymbol{V}_{\mathbf{UAV_O}}$ is the constrained UAV velocity output, also coined as the velocity command. The window for more advanced signal smoothing algorithms is open within this domain as a set of input/output vectors. Such algorithms can range from simple Linear Quadratic Regulator (LQR) controllers to more advanced Model Predictive Control (MPC) schemes. However, empirical tests show that the low pass filter utilized here is sufficient as a smoothing schema. In the event that more advanced smoothing techniques are required (such as when movement sensitive equipment are carried onboard the UAV), the option for replacing the low pass filter is a simple case of pick and place. The combination of completely decoupled systems which make up the entire OAN algorithm indicate its strength as a scalable solution for the challenge of OAN. Such scalability is often unheard of in more modern solutions employing Neural Networks or Fuzzy Logic, as parameters and algorithm design are locked in after training. Furthermore, the algorithm presented here is free of any form of training requirements, eliminating the need of acquiring a dataset before implementation.

UAV Stuck Detection

To detect scenarios where the UAV falls into a local minimum, a simple moving average velocity window is employed. Hidden in Equation (9) is an indicator that local minima exist within the global vector field. The behavior is described through:

$$\boldsymbol{u}_{\mathbf{UAV_T}} = \begin{cases} +\boldsymbol{u}_{\mathbf{UAV_C}} & D_{\text{prox}} < D \\ [0,0] & D_{\text{prox}} = D \\ -\boldsymbol{u}_{\mathbf{UAV_C}} & D_{\text{prox}} > D \end{cases} \tag{13}$$

Fundamentally, this point allows the technique listed here to be effective. The moving average window scrutinizes the average velocity, $|V|_{\text{ave}}$ over a fixed period of time t_{f}. When $|V|_{\text{ave}}$ falls below a predefined velocity threshold, a 'Drone is Stuck' (DIS) flag is triggered.

The equation for the moving average velocity window is as shown:

$$N = f \cdot t_{\mathrm{f}} \tag{14}$$

$$|V|_{\mathrm{ave}} = \frac{1}{t_{\mathrm{f}}} \cdot \int_{t-t_{\mathrm{f}}}^{t} |V| dt \tag{15}$$

$$= \frac{1}{N} \cdot \sum_{i-N}^{i} |V| \tag{16}$$

f represents the algorithm loop frequency. Equation (15) is the equation in continuous time form, while Equation (16) describes the window in discretized form. This solution is effective as the $\boldsymbol{V}_{\mathbf{UAV}} \rightarrow [0,0]$ as the UAV gets attracted to a local minimum on the global map. As is the trend of this algorithm, should something on a more sophisticated level be desired, the procedure is a simple drop in replacement.

METHODOLOGY

Phase 1 - Proof of Concept Evaluation

A simulation engine was built in MATLAB R2016b to allow comprehensive testing of the algorithm. The simulation engine results have been verified to replicate real world performance accurately. The simulation engine interface is akin to those shown in Figure 3, 5 and 6. Where black dots represent obstacles within the occupancy grid, and blue dots represent the path of the UAV. A two-phase performance evaluation is utilized in this study.

The first is a conceptual validation of the algorithm. This is done to verify that the algorithm functions as expected. Two maps are used for this, the first being a factory map derived off a real factory floor plan from [18]. The second map is a real estate map taken directly from an aerial image of a palm oil estate. The factory map is displayed in Figure 5, with the estate map shown in Figure 6.

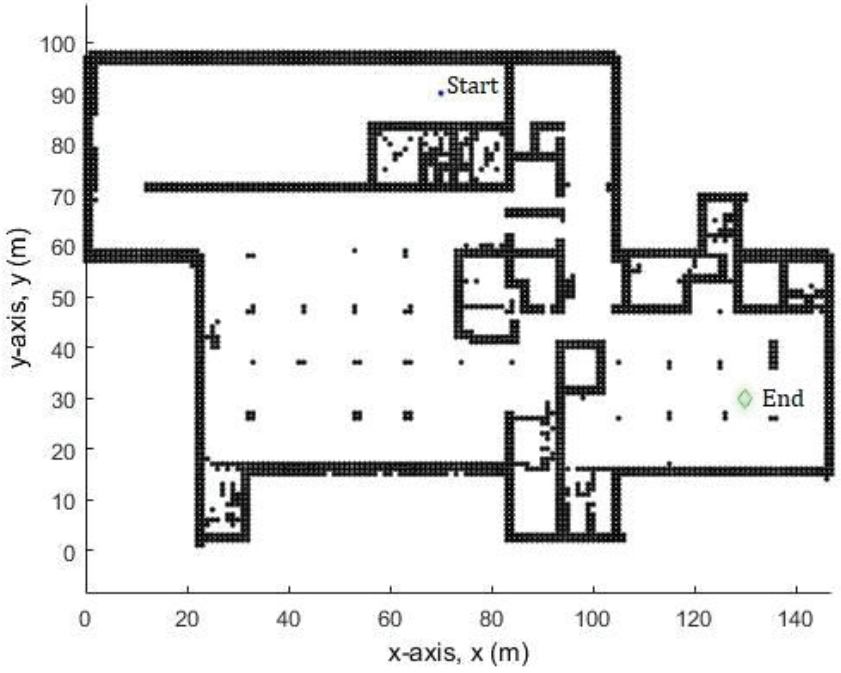
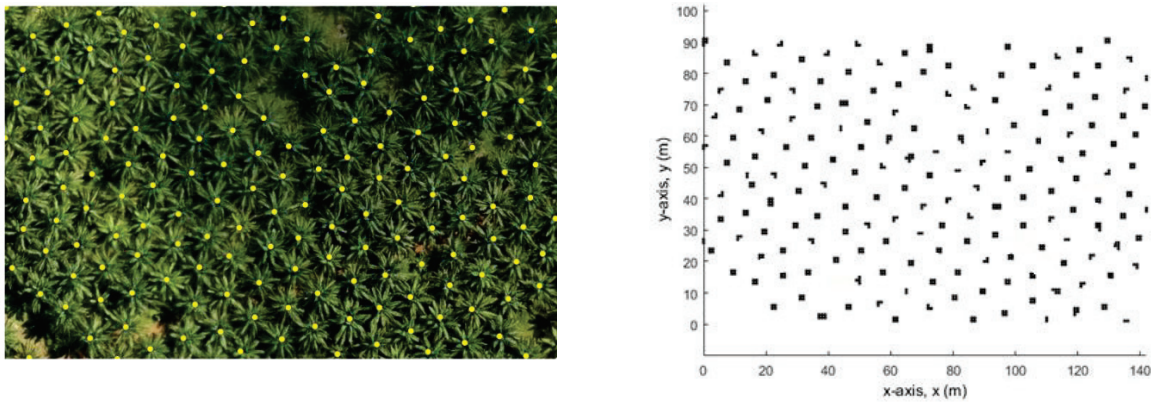


FIGURE 5. Factory map.



(a) (b)

FIGURE 6. (a) Aerial view of palm oil estate, with yellow dots depicting the individual palm oil trees. (b) Palm oil estate aerial view converted into a usable map for the simulation engine.

Phase 2 - Performance Evaluation

The second phase of performance evaluation involves comparison of the algorithm with other first-principle derived algorithms. Testing of different OAN algorithms is of difficulty due to the various forms of implementation available. This is further exaggerated by the different forms of simulation and validation engines used by various researchers, convoluting the testing of algorithms on different platforms. Of the international efforts to provide viable benchmark sets [16–18], not one has viably taken off. For this reason, COAA* will instead be compared to the main competing algorithm available – continuous A*. Other first-principle derived algorithms such as pure potential field methods or vector field methods do not guarantee convergence to global minimum and are hence not usable in this test. The performance metrics used in this test are computation cycles and time taken to completion. A total of 4 randomly generated maps are used during this phase, shown here in Figure 7.

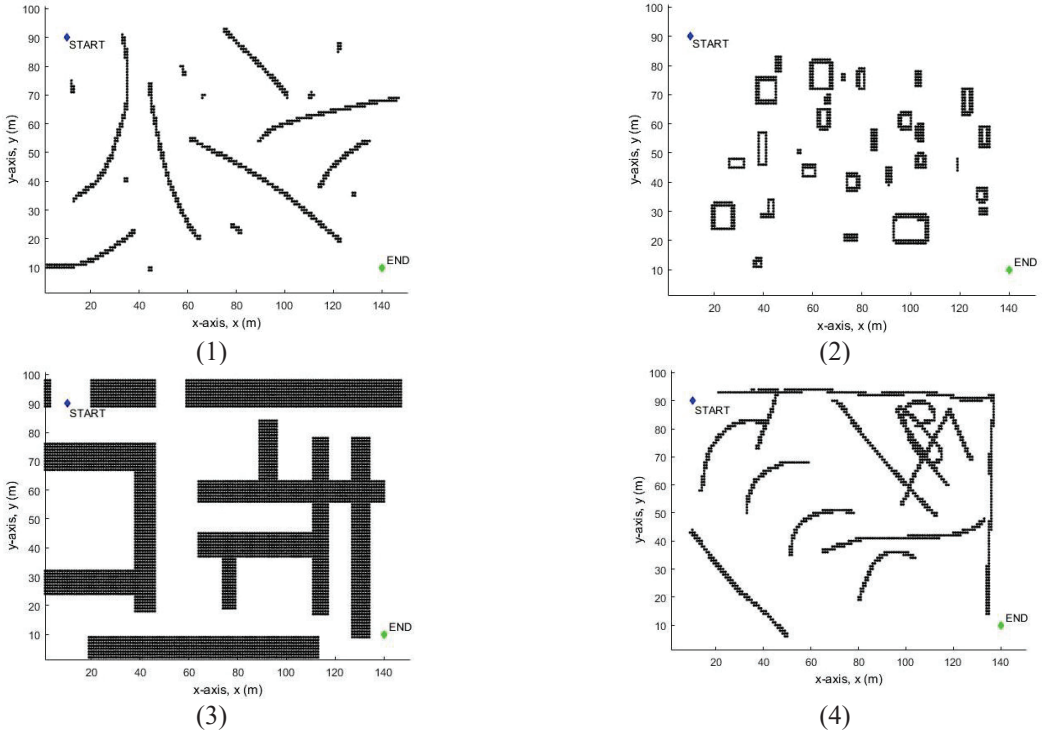
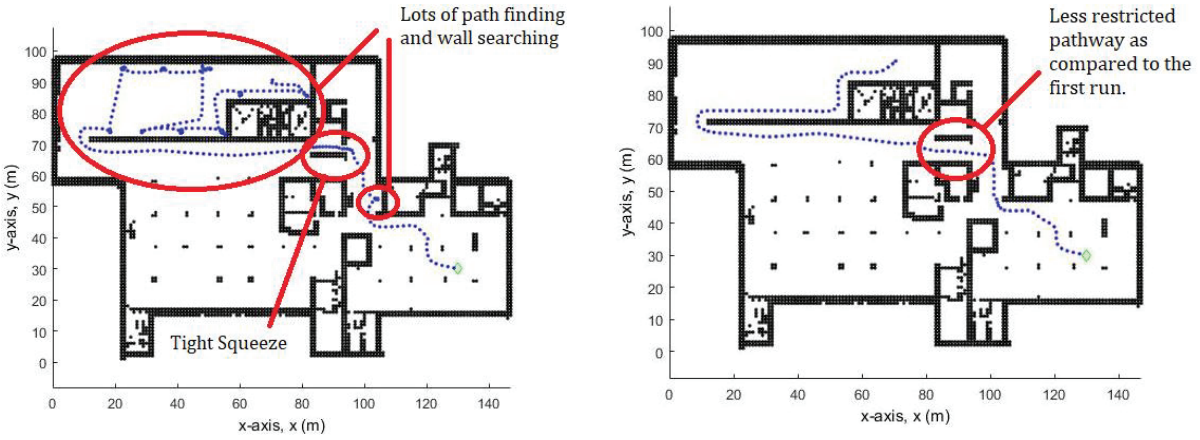


FIGURE 7. Four maps used to perform the performance evaluation of COAA* against other competing algorithms.

RESULTS AND DISCUSSION

Phase 1 - Proof of Concept Evaluation

The algorithm was constructed in the form of a MATLAB script, and a simulation engine was utilized to evaluate the performance of the algorithm in real world scenarios. As a preliminary, the performance of the algorithm is evaluated in two real world-based maps. The first map was run twice. This is to simulate the UAV operating within the same map space repeatedly, with each flight getting better as it learns the map space. The second map is intended to simulate cluttered outdoors environments. The task provided in this map was for the UAV to fly waypoint targeted paths, designed to simulate a UAV performing inventory keeping within the palm oil estate. The results for both maps are shown in Figure 8 and Figure 9.



(a) (b)

FIGURE 8. (a) First run in the factory map. Even without prior knowledge of the global map, the UAV is still able to reach the target point. (b) Second run in the factory map. The second run demonstrates a much smoother path, allowing the UAV to complete the map faster. Both these runs show that the UAV is able to find the most optimum path when possible (if plenty of information is available), and also able to adapt and learn of its surroundings when no information is provided.

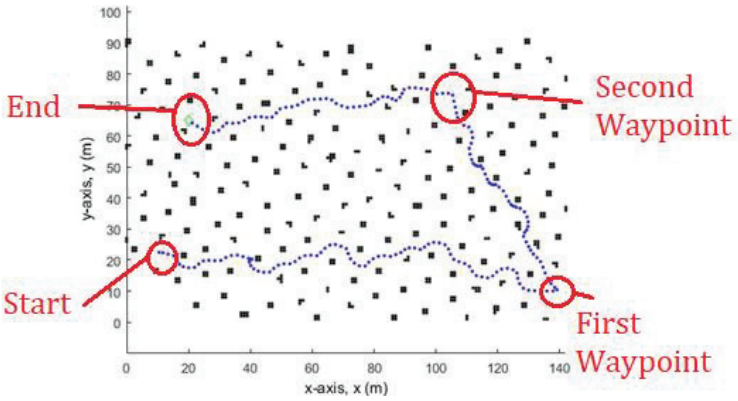


FIGURE 9. Simulated run of the OAN algorithm in a palm oil estate map with 2 waypoints before the end target. The algorithm is able to successfully avoid all obstacles and reach all waypoints, even when no information is provided. This run also demonstrates the efficacy of the obstacle avoidance implementation as A* was not executed at all, indicating that all trajectories were generated on-the-fly.

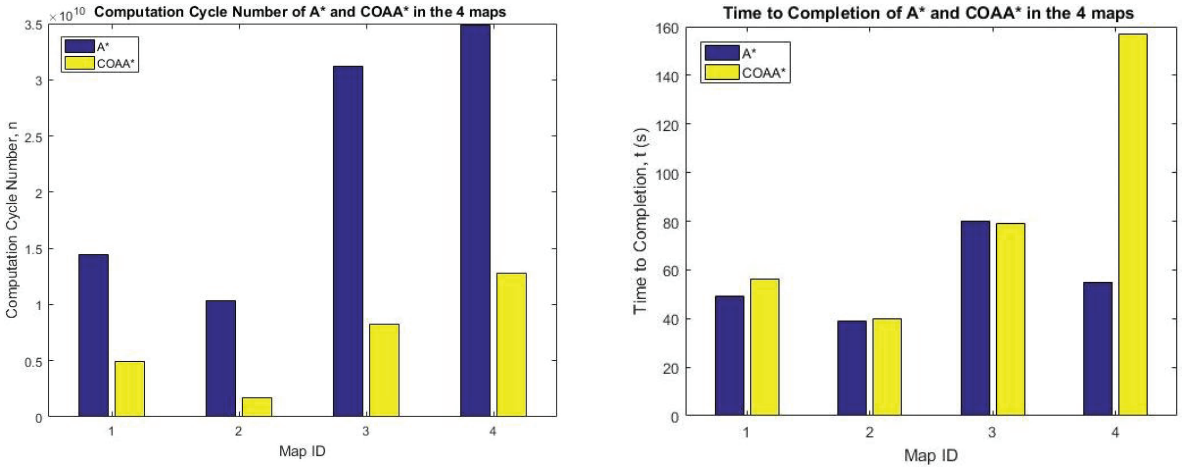
Phase 2 - Performance Evaluation

In this section, the performance of COAA* against continuous A* was done. The performance metrics used are computation cycle number and time to completion. For both implementations in this test, the list of hyperparameters used are shown in Table 1. Continuous A* was run with a loop time of 2 Hz.

TABLE 1. Hyperparameters used for the performance Evaluation of both algorithms

Hyperparameter	Value	Units
Observation Radius	10	Meter (m)
Proximity Allowance	3	Meter (m)
Maximum Velocity	5	Meters per Second (m/s)

The corresponding results for computation cycle number and time to completion for each of the four maps run under both these algorithms is illustrated in Figure 10. Computation cycle is the rough number of mathematical operations performed in order to complete the map. The time to completion metric is computed as total vehicle flight time, and time taken to perform computations is not taken into consideration. This is attributed that various UAV systems can have varying amounts of computing power available, which would affect the total time to completion if considered. Instead, the total flight time is taken instead, which depends on the ground truth of flight velocity being controlled.



(a) (b)

FIGURE 10. (a) Completion Cycle of A* and COAA* in the 4 maps. (b) Time to Completion of A* and COAA* in the 4 maps.

Figure 10(a) shows that the computational load of COAA* varies from a speed up factor of 3x to 10x. This is contributed through COAA* delaying computationally expensive operations for as long as possible. However, based on Figure 10(b), the results of COAA* is not always satisfactory shown by the time to completion of map 4. Throughout maps 1 through 3, COAA* takes a similar duration as A*, this is attributed to the both algorithms taking similar paths, shown in Figure 11(a)-(c). Unfortunately, in map 4, COAA* inadvertently takes a totally different route than A*, causing it to get stuck at multiple local minimums. This effect is a byproduct of the instinct-first search-second nature of COAA*. The corresponding paths are shown in Figure 11(a) through to Figure 11(d).

In reality, scenarios such as that in Figure 11(d) occur in more maze-like maps – such as the internals of an office building, where world structures are heavily concave. Such instances contain many local minimums, and is therefore desirable that A* be executed more as a countermeasure. However, if the UAV is intended to be operated in these scenarios, simply providing the UAV with more information of the map would bypass the drawbacks of COAA* immediately. If this is not possible, COAA* will just take a longer time to reach its destination, while still utilizing far less computation cycles. This is especially important for a UAV operated in these environments as the system would likely be much smaller, with lower payload capacity, and less available computational power, once again proving that COAA* is the suitable candidate for the task.

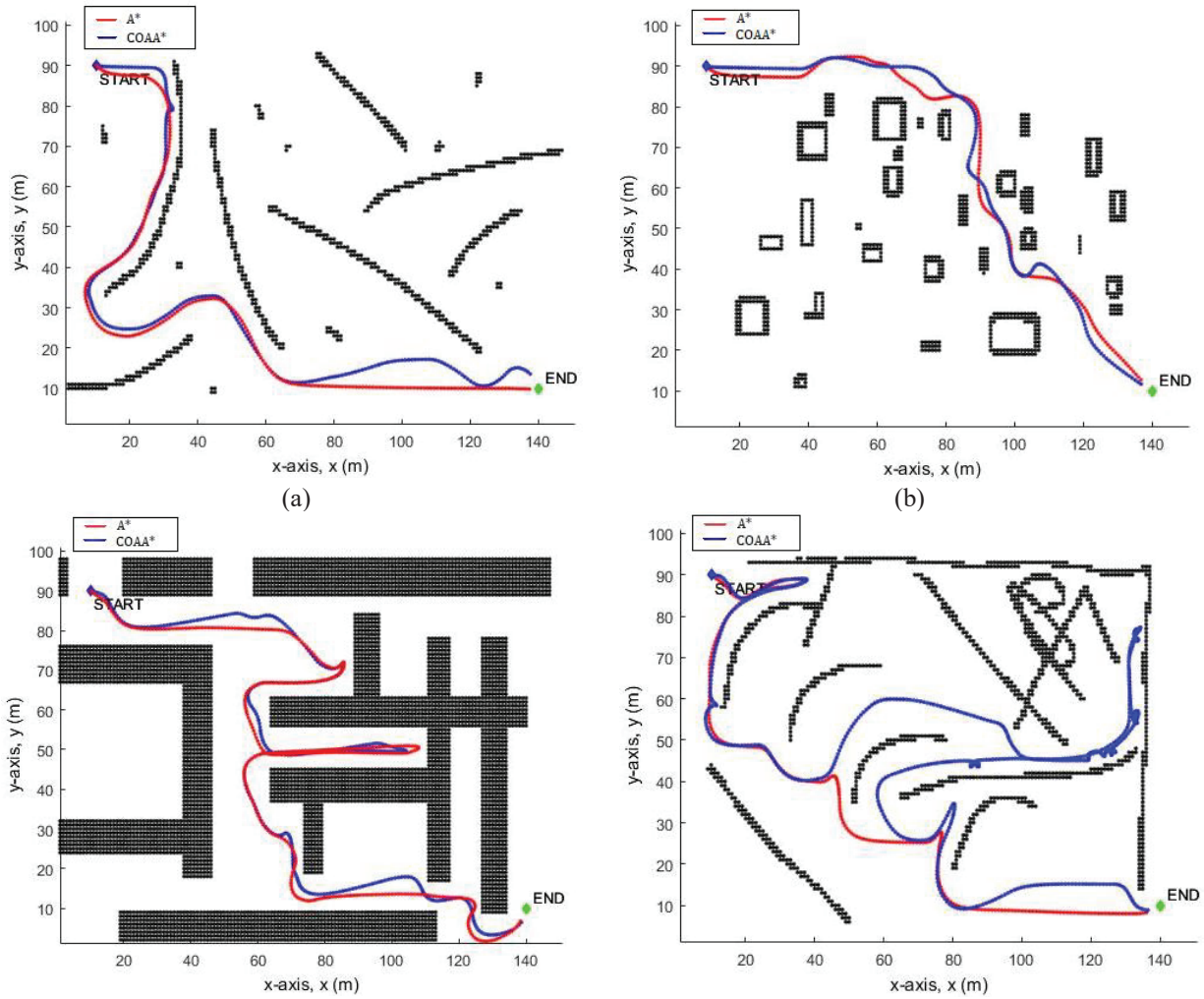


FIGURE 11. (a) (b) (c) Path taken by A* and COAA* for map 1, 2, and 3 respectively. (d) Path taken by A* and COAA* for map 4. COAA*'s performance falls behind continuous A* as it gets lost by taking a totally different path.

CONCLUSION

In this research work, COAA* was presented as an efficient, first-principles derived OAN algorithm. The algorithm presented here is superior compared to other first-principles derived algorithms in terms of robustness and computational efficiency, achieving up to a 10x speedup when compared to the most commonly employed navigation algorithm – continuous A*. The impact of such a technology is significant, as sectors which employ fully autonomous, standalone UAVs – search and rescue, delivery sectors, and more – can utilize COAA* as a launch-and-forget system, potentially much more economically. COAA* unfortunately falters as compared to continuous A* in scenarios where the map is totally unknown. And this is a sector of this work that is intended to be worked on in future studies. Better methods of triggering A* for COAA* is needed to enable performance matching to that of continuous A*. Furthermore, other techniques of speeding up the A* search such as Voronoi diagrams can be utilized.

REFERENCES

1. S. Nunna *et al.*, "Enabling real-time context-aware collaboration through 5G and mobile edge computing," in *12th International Conference on Information Technology-New Generations*, 2015, pp. 601–605.
2. S. Lange, N. Sunderhauf, and P. Protzel, "A vision based onboard approach for landing and position control of an autonomous multirotor UAV in GPS-denied environments," in *International Conference on Advanced Robotics (ICAR)*, 2009, pp. 1–6.
3. J. J. Tai, S. K. Phang, and C. L. Hoo, "Application of Steady-State Integral Proportional Integral Controller for Inner Dynamics Control Loop of Multi-rotor UAVs," in *2018 Fourth International Conference on Advances in Computing, Communication & Automation (ICACCA)*, 2018, pp. 1–6.
4. S. K. Phang, S. Z. Ahmed, and M. R. A. Hamid, "Design, Dynamics Modelling and Control of a H-Shape Multi-rotor System for Indoor Navigation," in *2019 1st International Conference on Unmanned Vehicle Systems (UVS)*, 2019, pp. 1–6.
5. S. K. Phang, S. Lai, F. Wang, M. Lan, and B. M. Chen, "Systems design and implementation with jerk-optimized trajectory generation for UAV calligraphy," *Mechatronics*, vol. 30, pp. 65–75, 2015.
6. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Autonomous Robot Vehicles*, Springer, 1986, pp. 396–404.
7. W. H. Huang, B. R. Fajen, J. R. Fink, and W. H. Warren, "Visual navigation and obstacle avoidance using a steering potential function," *Robotics and Autonomous Systems*, vol. 54, no. 4, pp. 288–299, 2006.
8. T. T. Mac, C. Copot, A. Hernandez, and R. De Keyser, "Improved potential field method for unknown obstacle avoidance using UAV in indoor environment," in *2016 IEEE 14th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 2016, pp. 345–350.
9. D. Fu-guang, J. Peng, B. Xin-qian, and W. Hong-Jian, "AUV local path planning based on virtual potential field," in *IEEE International Conference Mechatronics and Automation*, 2005, vol. 4, pp. 1711–1716.
10. R. S. Pol and M. Murugan, "A review on indoor human aware autonomous mobile robot navigation through a dynamic environment survey of different path planning algorithm and methods," in *2015 International Conference on Industrial Instrumentation and Control (ICIC)*, 2015, pp. 1339–1344.
11. M. Guerra, D. Efimov, G. Zheng, and W. Perruquetti, "Avoiding local minima in the potential field method using input-to-state stability," *Control Engineering Practice*, vol. 55, pp. 174–184, 2016.
12. D. Panagou, "Motion planning and collision avoidance using navigation vector fields," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 2513–2518.
13. H. Chen, K. Chang, and C. S. Agate, "UAV path planning with tangent-plus-Lyapunov vector field guidance and obstacle avoidance," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 2, pp. 840–856, 2013.
14. F. Abdessemed *et al.*, "A hierarchical fuzzy control design for indoor mobile robot," *International Journal of Advanced Robotic Systems*, vol. 11, no. 3, p. 33, 2014.
15. E. Ayari, S. Hadouaj, and K. Ghedira, "A fuzzy logic method for autonomous robot navigation in dynamic and uncertain environment composed with complex traps," in *2010 Fifth International Multi-conference on Computing in the Global Information Technology*, 2010, pp. 18–23.
16. A. Bakdi, A. Hentout, H. Boutami, A. Maoudj, O. Hachour, and B. Bouzouia, "Optimal path planning and execution for mobile robots using genetic algorithm and adaptive fuzzy-logic control," *Robotics and Autonomous Systems*, vol. 89, pp. 95–109, 2017.
17. A. H. Karami and M. Hasanzadeh, "An adaptive genetic algorithm for robot motion planning in 2D complex environments," *Computers & Electrical Engineering*, vol. 43, pp. 317–329, 2015.
18. P. Fletcher, "Founding & requirements of starting a plastics product factory: Case Hämeen Lanka," 2013.
19. W. Nowak, A. Zakharov, S. Blumenthal, and E. Prassler, "Benchmarks for mobile manipulation and robust obstacle avoidance and navigation," *Best Practice in Robotics Deliverable 3.1*, 2010.
20. D. Calisi, L. Iocchi, and D. Nardi, "A unified benchmark framework for autonomous mobile robots and vehicles motion algorithms (MoVeMA benchmarks)," in *Workshop on experimental methodology and benchmarking in robotics research, Robotics Science and Research Conference*, 2008.
21. N. D. M. Ceballos, J. A. Valencia, and N. L. Ospina, *Quantitative performance metrics for mobile robots navigation*. INTECH Open Access Publisher, 2010.