Unmanned Systems, Vol. 0, No. 0 (2013) 1–16

COAA* - An Optimized Obstacle Avoidance and Navigational Algorithm for UAVs Operating in Partially Observable 2D Environments

Jun Jet Tai[a], Swee King Phang[b,*], Felicia Yen Myan Wong[b]

[a] *Institute for Future Transport and Cities, Coventry University, Priory St, Coventry CV1 5FB, United Kingdom*

[b] *School of Computer Sciences and Engineering, Taylor's University, 1, Jalan Taylors, 47500 Subang Jaya, Selangor, Malaysia*

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

Obstacle avoidance and navigation (OAN) algorithms typically employ offline or online methods. The former is fast but requires knowledge of a global map, while the latter is usually more computationally heavy in explicit solution methods, or is lacking in configurability in the form of artificial intelligence (AI) enabled agents. In order for OAN algorithms to be brought to mass produced robots, more specifically for multirotor unmanned aerial vehicles (UAVs), the computational requirement of these algorithms must be brought low enough such that its computation can be done entirely onboard a companion computer, while being flexible enough to function without a prior map, as is the case of most real life scenarios. In this paper, a highly configurable algorithm, dubbed Closest Obstacle Avoidance and A* (COAA*), that is lightweight enough to run on the companion computer of the UAV is proposed. This algorithm frees up from the conventional drawbacks of offline and online OAN algorithms, while having guaranteed convergence to a global minimum. The algorithms has been successfully implemented on the Heavy Lift Experimental (HLX) UAV of the Autonomous Robots Research Cluster in Taylor's University, and the simulated results match the real results sufficiently to show that the algorithm has potential for widespread implementation.

Keywords: Obstacle avoidance; navigation; multirotor; unmanned aerial vehicle

1. Introduction

Unmanned Aerial Vehicles (UAVs), commonly known under the encompassing term of 'drones', are aerial robots that are operated remotely, with or without a ground pilot. One of the more popular forms of UAVs are of the multirotor variant, where a set of four, six, eight or more propulsion systems are arranged to allow the aircraft to make very precise maneuvres in 3D space [1]. Coupled with payload manipulation, these systems end up being very capable mobile robots in both structured and unstructured environments [2, 3]. Due to the low-cost and scalable design, UAVs are very valuable tools in many industries not limited to photography [4], cinematography [5], agriculture [6], and security sectors [7].

The main attraction of modern multirotor systems are the ability to traverse over large open areas using Global Positioning System (GPS) enabled waypoint flight. For the most part, these systems perform fine in open-air, where GPS signal is readily available. However, with the advent of many new technologies such as 5G networking [8], AI powered vision processing [9], scalable information frameworks through blockchain-based Internet-of-Things (IoT), as well as advanced UAV control algorithms [10], the avenue for UAVs to operate in closed, partially observable environments are opening up. This is especially true in closed urban landscapes such as Kuala Lumpur, Tokyo, or New York, as well as indoor environments like factory floors or within an office building. In these scenarios, tasks such as same-building delivery, automated search and rescue, automated inventory keeping, or city-level courier can be performed autonomously with a UAV, alleviating the need for humans to be in the loop of the system. For this reason, fully autonomous OAN enabled UAVs are a form of technology that is actively pursued by many parties.

There are two primary classifications for OAN algorithms, offline and online. Offline methods generally revolve around the staple path finding method of A* and operate on a known global map. The global map is first broken up into a series of nodes, and a path from the beginning to the end is formulated. In the case of UAV flight, multiple forms of smoothing functions are then applied over this path to form a path that can be feasibly flown by a multirotor aircraft (to avoid having areas with impossible flight dynamics). Smoothing functions are generally computationally heavy, and hence done offboard on a workstation PC [11]. This reason, coupled with the need of a known global map, disallows offline methods to be readily

used on a resource constrained platform such as a UAV in a partially observable environment.

By contrast, online OAN algorithms are computationally much simpler, utilizing a series of basic rules or equations to achieve obstacle avoidance while actively heading towards a target. The simplest form of obstacle avoidance algorithms are derivatives of the famous potential field method [12]. Examples of this can be seen in unicycle style robots [13], UAVs [14] and other high speed vehicles [15]. However, these methods do not guarantee global convergence to a target, some may even cause convergence to occur at a location far from the intended target. Furthermore, the inherent inability to produce quickly curving trajectories around any object is a flaw of many quickly computable potential field. As a direct response to this, vector fields are also used for obstacle avoidance [16, 17]. These algorithms are capable of designing flight paths that consider the conservation of flight velocity vector, allowing the UAV to take smooth, curved motions around obstacles. Nevertheless, the same pitfalls of potential field method still applies to vector field methods, that the convergence of the robot to the global minima is not guaranteed. A good reference for existing heuristically driven and exact OAN algorithms can be studied in [18].

By modern standards, pure potential field or vector field methods are almost never used in a standalone fashion due to the many downsides inherent to these methods. Instead, the current trend in OAN seems to favour the use of deep learning methods [19, 20, 21]. Broadly speaking, these algorithms typically express the task of OAN as a highly parameterized deep neural network with sensory inputs and actuator outputs. The goal of this deep neural network is to solve the task of OAN through the optimization of a surrogate scalar reward function. This deep neural network is then made to solve OAN tasks in simulation on a scale of 1e6 or more trajectories before a convergent solution is formed. Because of the flexible and highly configurable architectures of deep neural networks and reward function shaping, almost any form of OAN solving behaviour can be obtained given a sufficient number of trajectories from which the algorithm can learn from [22]. For UAVs, some of these include danger-aware OAN algorithms in a multi agent setting [23] and behaviour-based or hierarchical reinforcement learning agents [24].

While there is no doubt that deep learning based methods are powerful, they inherently do not provide the required scalability required for a unified OAN framework - one that can be utilized across any UAV platform, easily modifiable through changing of user intuitive variables. For example, to change the minimum distance the UAV should maintain between itself and any obstacle, it is not exactly trivial on how one should do this when given a deep neural network and its weights. Even if the behaviour of the network can be tweaked through the reward function, this would most often require a retraining of the network from scratch, limiting its deployability when hardware configurations are changed. In contrast to these methods, this paper introduces a heuristically designed algorithm dubbed Closest Obstacle Avoidance and A* (COAA*) with several notable advantages over existing OAN algorithms. More specifically, the algorithm here introduces these benefits:

- The algorithm guarantees convergence to a global minimum.
- The algorithm is specifically designed for UAV based platforms, taking into consideration the unique flight dynamics specific to multirotor platforms.
- The algorithm only requires a partially observable environment to operate in, and no prior offline map is needed for the algorithm to work. It is assumed that the UAV is able to only detect obstacles within a certain region of observability. In addition, the algorithm can comfortably operate in non-constant environments, and can be easily modified to operate in dynamic ones.

2. Hardware Setup

In general, COAA* can operate on most multirotor UAV platforms with decent companion computer and flight management unit (FMU) pair. As for the constraint for this research work, the algorithm is targeted towards running on systems with less than 5 kg takeoff weight while the computational power must be similar to that of a Raspberry Pi 3B or better. To realize the project, implementation of COAA* was done on an Intel UP Squared single board computer paired to a Pixhawk 4 FMU. The UAV platform used for the project is a custom-built TAROT 650 class multirotor known internally as the Heavy Lift Experimental (HLX). The HLX, Pixhawk 4, and Intel UP Squared are shown in Fig. 1.

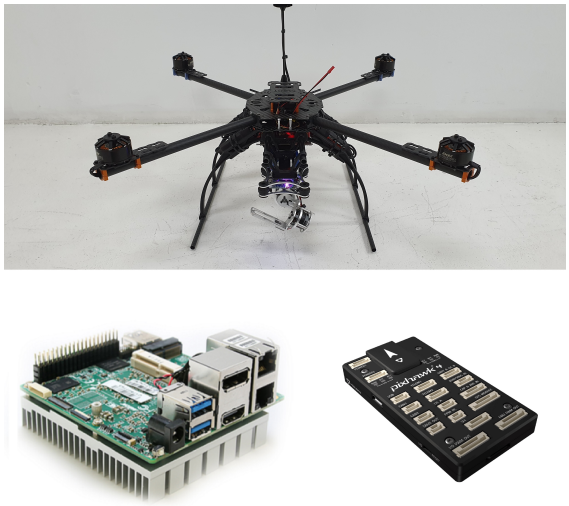

Fig. 1. Top: HLX platform; Bottom Left: Intel UP Squared single board computer; Bottom Right: Pixhawk 4 FMU

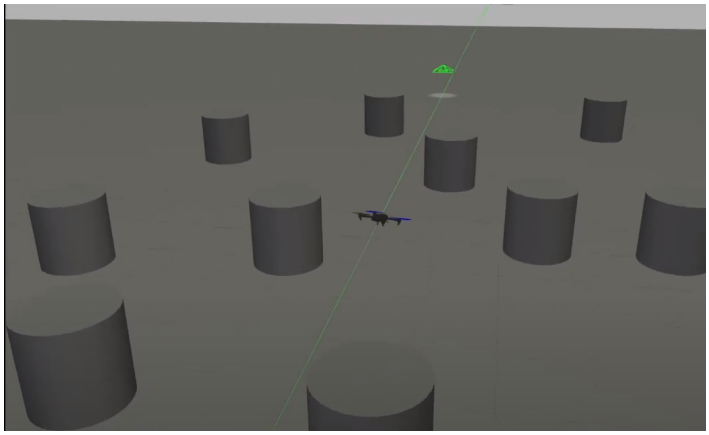

Fig. 2. Gazebo environment used to safely sandbox the firmware before being used on the actual UAV.

To realize accurate localization and tracking, a simple visual-based localization and mapping system fused with GPS sensor data is used to inform the UAV of local obstacles while allowing it accurate localization through a standard off-the-shelf Extended Kalman Filter. Documentation of the visual localization and mapping algorithm has been previously documented [25]. As the mapping and localization of the UAV is not part of this research, these components will not be covered in this manuscript. A custom UAV flight control system was devised for the project, and this is detailed in Section 3.

Prior to actual flight testing on the UAV, extensive testing and simulation was iteratively developed in MATLAB, before porting the algorithm to C++ using the MAVROS library in a Linux environment (Ubuntu 18.04.02 on a PC, Ubuntu Mate on the companion computer). A simulated environment for visualization is also built in the GAZEBO simulation environment, shown in Fig. 2.

3. UAV Control System

For OAN algorithms, good control regimes are paramount to system performance. Otherwise, there will be no foundation from which the trajectory planning algorithm to build on. For our implementation, a custom steady state integral based controller is used for UAV attitude control. This implementation is inspired from motor control regimes and has also been previously employed on UAVs [26, 27, 10, 28].

We employ the generic two stage control loop intrinsic to most UAV controllers, shown here in Fig. 3. The inputs to the system include the position $\boldsymbol{p_r}$, velocity $\boldsymbol{v_r}$, and acceleration $\boldsymbol{a_r}$ reference points relative to the global frame. The outer-loop controller is then responsible for representing the inputs as a global frame acceleration reference point $\boldsymbol{a_g}$. A simple rotation matrix converts this reference to the UAV body frame $\boldsymbol{a_b}$. Thereafter, the inner loop controller is responsible for providing the indvidual throttle u_{thr}, aileron u_{ail}, elevator u_{ele}, and rudder u_{rud} commands to the UAV. These commands are converted to individual motor commands through a mapping matrix before being sent to the individual speed controllers. Basic feedback is utilized to make this a closed-loop system.

In our implementation, the outer-loop controller is a generic PI controller, implemented as an off-the-shelf module available in the PX4FMU multirotor firmware. However, to allow the UAV to track the reference point for acceleration, the steady-state integral proportional integral controller (SIPIC) is applied. This solution allows tuning of the UAV to be eliminated, and replaced with simple trifilar pendulum tests to obtain the UAVs moment of inertia matrix as well as motor tests to obtain motor thrust and moment coefficients. For sake of completeness, the block diagram of the inner loop controller is shown in Fig. 4. For a more formal derivation, the reader is directed toward its derivation [10].

4. Algorithm Architecture

To lay the groundwork, assume the UAV has partial observability of its environment, where it is able to perceive obstacles within a certain radius R_{OBS}. Let $M_G(x, y) \in \{0, 1\}^{(m\times n)}$ denote the ground truth occupancy grid of a 2D global map of size $m \times n$. The map known to the UAV itself is denoted as $M_L(x, y) \in \{0, 1\}^{(m\times n)}$ and starts off as all zeroes. As the UAV explores more of its environment, M_L is updated to more accurately reflect M_G. Therefore, as $t \to \infty$, $M_L \to M_G$. With slight abuse of notation, this describes the ideal case of all simultaneous localization and mapping (SLAM) algorithms [14, 29]. Additionally, issues of inaccurate odometrical data $O(x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}, ...)$ and potential of failed loop closures is not a concern of this study. Given the robustness of modern day perception and estimation techniques of off the shelf FMU systems, this is very unlikely to pose any issue to the study. Regardless, the design of the algorithm will be built to accomodate some form of noisy data, subject to reasonable constraints.

For the purpose of self-preservation, we consider gaps between obstacles of size less than that a multiple of the size of the UAV itself, αl_{UAV} as non-traversable, where $\alpha \in \mathbb{R}$. The purpose of this is twofold. Firstly, this avoids putting the UAV in any possibly dangerous position as multirotor systems can suffer from severe prop wash in closed quarter environments. Secondly, this puts a floor to the minimum recommended size of the occupancy grid used by the UAV. Should the UAV be of maximum width $l_{UAV} = 1$, a map resolution of approximately 0.5 is sufficient to provide enough resolution for the algorithm to perform well. This keeps the storage of the local map in the system memory efficient, even without data compression, a one kilometer squared area will only take up approximately 4 MBytes.

A portion of this section will be illustrated using simulation images taken from MATLAB. For purpose of readibility, the map legend will be described here with reference to Fig. 5.

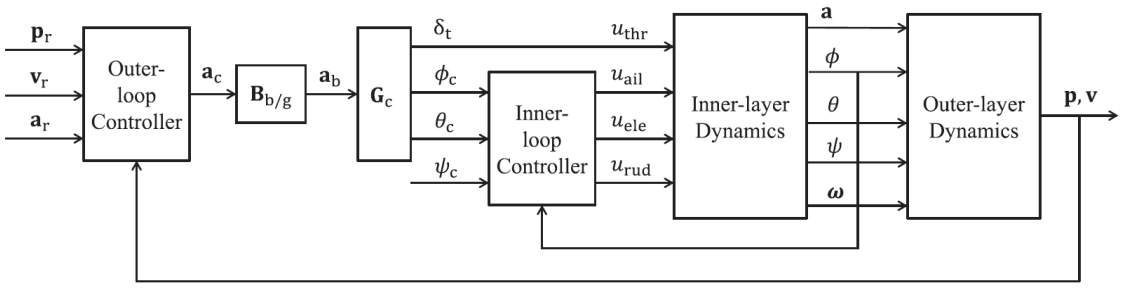

Fig. 3. Two stage control for multirotor UAVs.

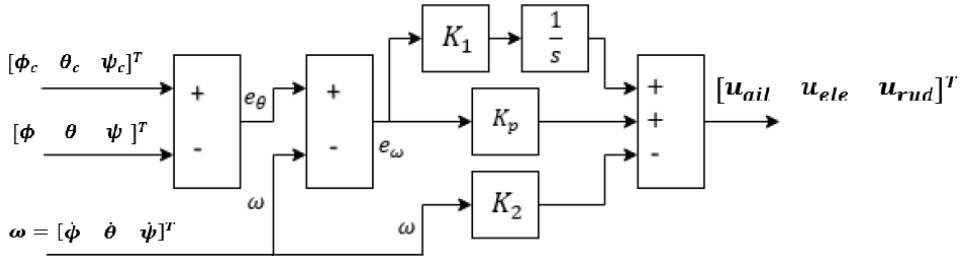

Fig. 4. SIPIC controller used as an inner-loop controller in the UAV.

In the map:-

- x and y axes represent the position of any entity in the map.
- The target location is denoted with the green diamond with the label *Target*.
- Red filled circles represent obstacles in the global map, M_G that is reflected in the local map, M_L.
- Red unfilled circles represent obstacles in the global map, M_G that is not reflected in the local map, M_L, ie: undiscovered obstacles.
- Blue circles represent the position of the UAV in 1 second intervals.
- Blue arrows represent the heading of the UAV, $\overrightarrow{u}_{\mathbf{UAV}} \in \mathbb{R}^2$
- Dark grey lines represent the closest obstacle to the drone at a time t, while lighter grey arrows represent the second closest obstacle to the drone at time t.

The simulation engine of the drone is a modified one used in [30] derived from Newton-Euler equations [31, 32, 33]. The simulation engine is built such that it accepts a velocity reference as input.

Prior to describing the algorithm in specific, the overall architecture of the algorithm is described in Fig. 6. The overarching strategy of COAA* is to prioritize fast and efficient computation everywhere, and only utilize the computationally heavy A* algorithm when necessary. The main operation regime of the algorithm uses a hybrid localized vector field and potential field method, and the overall path of the UAV is guided by a non-linear cost path finding algorithm that prioritizes free and open area flight.

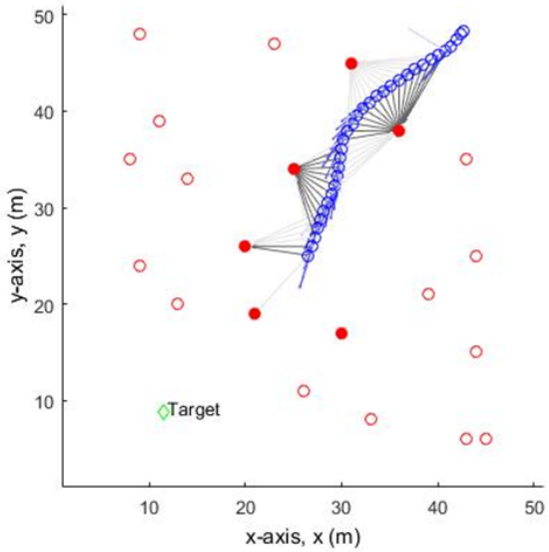

Fig. 5. Visual map used in the the MATLAB simulation.

4.1. *Vector Field and Potential Field Blending*

The main operating of regime of the algorithm is reliant on a hybrid vector and potential field formulation. Firstly, all vectoral notations are in $\mathbb{R}^2$. Let $\boldsymbol{P}_{\mathbf{T}}$ denote a target

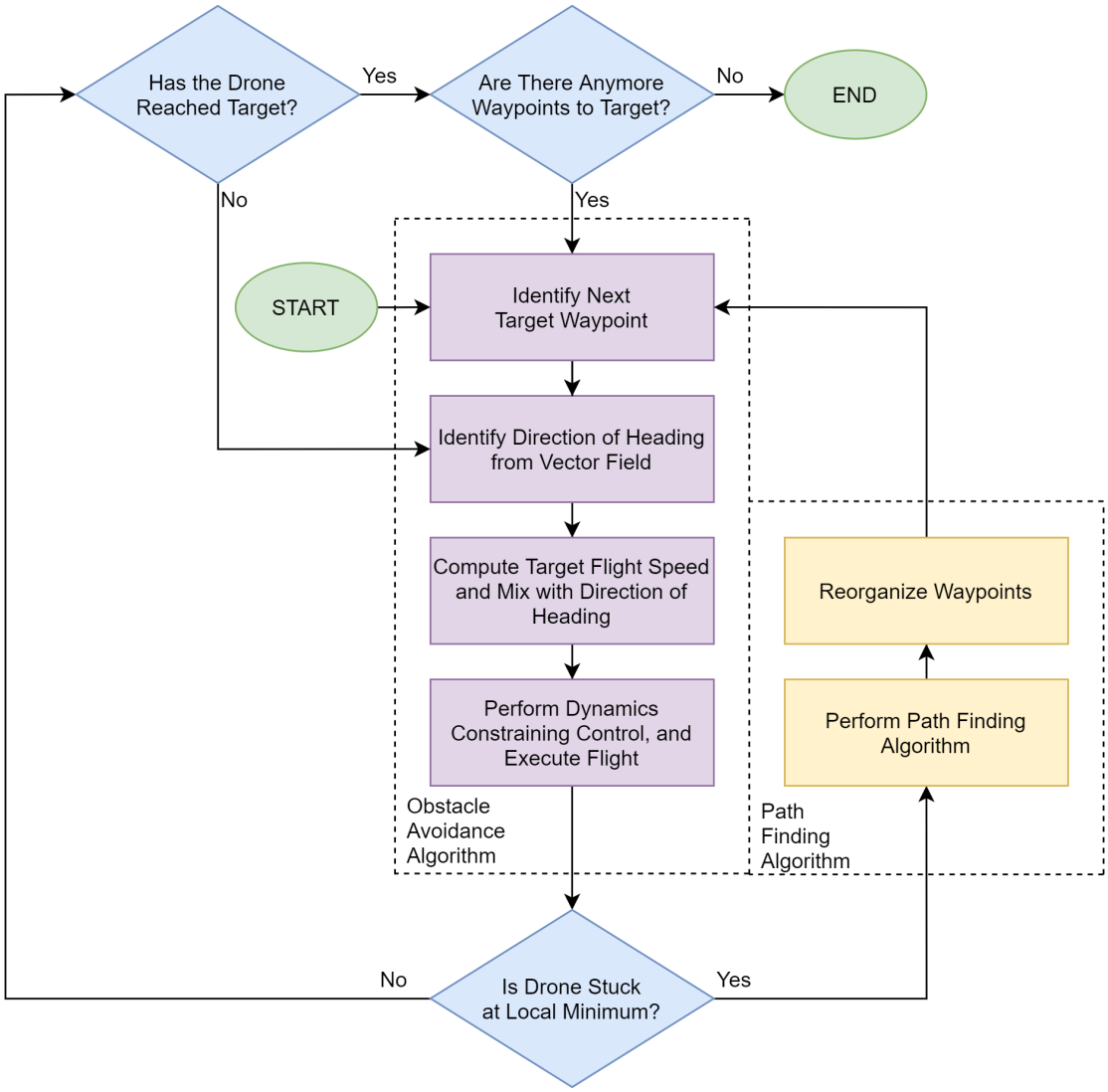

Fig. 6. Overarching algorithm state architecture.

waypoint, and $\boldsymbol{P_P}$ denote the position of the drone. The algorithm is designed to always look for the two closest filled positions in the occupancy grid (loosely, these can be thought of as being obstacles on the map). Let the position of these two obstacles be $\mathbf{O_i}$ where $i \in \{1, 2\}$. The reference heading of the UAV can then be computed through a series of vector computations:

$$\boldsymbol{u}_{\mathbf{O2T_i}} = \frac{\boldsymbol{P_T} - \boldsymbol{P}_{\mathbf{O_i}}}{|\boldsymbol{P_T} - \boldsymbol{P}_{\mathbf{O_i}}|} \tag{1}$$

$$\boldsymbol{u}_{\mathbf{P2O_i}} = \frac{\boldsymbol{P}_{\mathbf{O_i}} - \boldsymbol{P_P}}{|\boldsymbol{P}_{\mathbf{O_i}} - \boldsymbol{P_P}|} \tag{2}$$

$$\boldsymbol{u}_{\mathbf{T2P}} = \frac{\boldsymbol{P_T} - \boldsymbol{P_P}}{|\boldsymbol{P_T} - \boldsymbol{P_P}|} \tag{3}$$

Intuitively, each vector can be thought of as a vector of position-to-position of unitary length. Thus, the subscript $\mathbf{O2T_i}$ can be read as 'obstacle-to-target for obstacle i'. Correspondingly, subscripts $\mathbf{P2O_i}$ and $\mathbf{T2P}$ are read as 'drone position-to-obstacle for obstacle i' and 'target-to-position for the position of the drone'. As stated above, $i \in \{1, 2\}$ for the two closest obstacles to the current drone position. The vectors are used to compute the angle and vectors:

$$\begin{aligned}\theta_i = &- \operatorname{atan2}(\boldsymbol{u}_{\mathbf{P2O_{i,x}}}, \boldsymbol{u}_{\mathbf{P2O_{i,y}}}) \\ &+ \operatorname{atan2}(\boldsymbol{u}_{\mathbf{O2T_{i,x}}}, \boldsymbol{u}_{\mathbf{O2T_{i,y}}})\end{aligned} \tag{4}$$

$$\boldsymbol{u}_{\mathbf{UAV_i}} = \begin{cases} \begin{bmatrix} \boldsymbol{u}_{\mathbf{P2O_{i,y}}} \\ -\boldsymbol{u}_{\mathbf{P2O_{i,x}}} \end{bmatrix}, & \text{if } 0 < \theta < \frac{\pi}{2} \\ \begin{bmatrix} -\boldsymbol{u}_{\mathbf{P2O_{i,y}}} \\ \boldsymbol{u}_{\mathbf{P2O_{i,x}}} \end{bmatrix}, & \text{if } -\frac{\pi}{2} < \theta < 0 \\ \begin{bmatrix} \boldsymbol{u}_{\mathbf{T2P_x}} \\ \boldsymbol{u}_{\mathbf{T2P_y}} \end{bmatrix}, & \text{otherwise} \end{cases} \tag{5}$$

θ_i and $\boldsymbol{u}_{\mathbf{UAV_i}}$ are associated to obstacles $i \in 1, 2$. Finally, the reference heading is computed through:

$$\boldsymbol{u}_{\mathbf{UAV_C}} = \frac{\boldsymbol{u}_{\mathbf{UAV_1}} \cdot D_2 + \boldsymbol{u}_{\mathbf{UAV_2}} \cdot D_1}{|\boldsymbol{u}_{\mathbf{UAV_1}} \cdot D_2 + \boldsymbol{u}_{\mathbf{UAV_2}} \cdot D_1|} \tag{6}$$

where D_1 and D_2 are simply the Euclidean distance from $\boldsymbol{P_P}$ to $\mathbf{O_1}$ and $\mathbf{O_2}$ respectively.

Graphically, Eq. 1 to Eq. 7 form the individual vector fields around each obstacle as shown in Fig. 7. Eq. 6 simply performs a bi-linear interpolation of headings for the two closest obstacles to the UAV. Note that Eq. 1 to Eq. 6 will only be computed when there are obstacles within a certain radius of the UAV, $D_1 < D_{infl}$ and $D_2 < D_{infl}$. Consequently, when there is only one obstacle within D_{infl}, we simply set $\boldsymbol{u}_{\mathbf{UAV_C}} = \boldsymbol{u}_{\mathbf{UAV_1}}$.

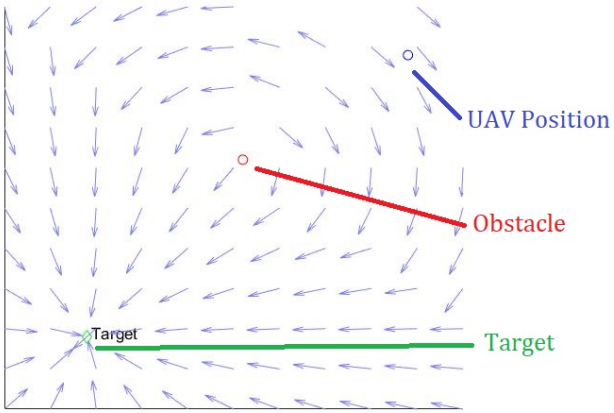

Fig. 7. Vector field formed around each obstacle representing reference velocity heading.

To prevent the UAV from flying through gaps smaller than D_{prox}, we simply do:

$$\lambda = \begin{cases} \frac{D_{\text{prox}}}{D_1}, & \text{if } D < D_{\text{prox}} \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

$$\boldsymbol{u}_{\mathbf{UAV_T}} = \frac{\boldsymbol{u}_{\mathbf{UAV_C}} - \lambda \cdot \boldsymbol{u}_{\mathbf{UAV_C}}}{|\boldsymbol{u}_{\mathbf{UAV_C}} - \lambda \cdot \boldsymbol{u}_{\mathbf{UAV_C}}|} \tag{8}$$

λ is simply a linear scaling factor which flips the heading vector. Eq. 7 and Eq. 8 form the conditional:

$$\frac{\boldsymbol{u}_{\mathbf{UAV_T}}}{\boldsymbol{u}_{\mathbf{UAV_C}}} = \begin{cases} 1, & \text{if } D_{\text{prox}} < D \\ 0, & \text{if } D_{\text{prox}} = D \\ -1, & \text{if } D_{\text{prox}} > D \end{cases} \tag{9}$$

To determine the magnitude of velocity that the UAV should follow, any continuous function of D_i can be used. In our implementation, we chose the Sigmoid function, represented in Fig. 8 and has the following formula:

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

such that the final velocity reference of the UAV is:

$$\boldsymbol{V}_{\mathbf{UAV_T}} = \boldsymbol{u}_{\mathbf{UAV_T}} \cdot V_a \tag{11}$$

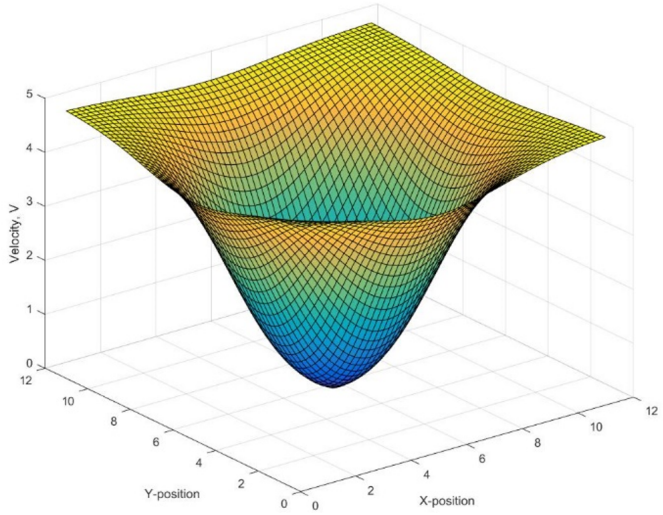

Fig. 8. Visual potential field around an obstacle centered at [5, 5].

The formulation given here is novel compared to other works. The primary advantage is that its formulation is explicit and does not depend on numerical solvers, allowing it to be very lightweight.

To compensate for any discontinuities in the flight velocity profile, the velocity command is first computed through a first order low pass filter. This filter is represented in the Laplace domain as:

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

$\boldsymbol{V}_{\mathbf{UAV_O}}$ is the velocity command fed to the UAV FMU. Note that the opportunity for more advanced signal smoothing algorithms to be applied is readily available as this is but a simple single-input-single-output (SISO) system. In the event that motion sensitive equipment is carried onboard the UAV, switching the smoothing function is a simple pick and place operation. This further highlights the strengths of COAA* as a scalable algorithm. This is in stark constrast to Neural Network or Fuzzy Logic based

control, where the operational parameters of the algorithm are encoded within the weights of the network, uninterpretable to human designers.

4.2. *A* Search Integration*

Up to this extent, no notion of how the algorithm will eliminate the presence of local minimum is detailed. This will be the main goal of this section. Two challenges arise. The first involves allowing the UAV to distinguish points in the map where it has reached a local minimum. The second involves dealing with the local minimum directly. There are several methods to solve the first challenge. This ranges from applying look-ahead searches to identify local minimums to simply applying a moving average window to the UAV's velocity profile. In the name of simplicity, we chose the latter, given by the following equation:

$$V_{ave} = \frac{1}{t_f} \cdot \int_{t-t_f}^{t} |\boldsymbol{V}_{\mathbf{UAV_O}}| dt \tag{13}$$

or in the discrete domain:

$$V_{ave} = \frac{1}{N} \cdot \sum_{i-N}^{i} |\boldsymbol{V}_{\mathbf{UAV_O}}| \tag{14}$$

As soon as V_{ave} goes below a certain threshold, we identify that the UAV has gotten stuck at a local minimum. This brings us to the second issue.

Graph search techniques are the core to path finding challenges. Therefore, we employ a modified A* algorithm which includes a herein called 'danger' cost. Interested readers are encouraged to explore the its formulation [34]. This cost denotes the proximity of connected paths to the obstacles around it. This allows the UAV to prioritize wide open spaces where it is able to fly at higher flight speeds with minimal risk. Considering that A* operates on a map, we simply apply the algorithm to the local map, M_L at the time of inference. Since A* is a computationally expensive algorithm to run, it is only re-executed when required. Intrinsically, this allows the algorithm to continually run with more information each time as the UAV traverses more and more of the map. By this analogy, as $t \to \infty$, the UAV is bound to reach the global minimum.

4.3. *Waypoint Generation*

While the A* algorithm works to generate a flight path, the vector and potential field algorithm operates on a series of waypoints. Although formulating the path as a chain of many waypoints is possible, this is not efficient and renders the algorithm to be unrobust to the appearance of smaller obstacles on the generated flight path of the UAV. To circumvent this, the path generated by the A* algorithm is fed through a split-and-merge (SaM) algorithm. At its core, what a SaM does is iteratively segment a connected path by evaluating the summed error between a waypoint-based path and the raw path given by the A* algorithm. Formally, the steps to the SaM is used here is illustrated in Fig. 9 and is sequenced as follows:

(1) Put the first and last point from the path, $P = \{A, ..., B\}$ into a list of waypoints, connecting the points A and B with a waypoints list, $WP = \{A, B\}$.
(2) Form a linear equation of $Y = m_1 X + C_1$ between two connected points, where the subscript 1 depicts the m and c values for the equation formed for the first line, which for the first step is between A and B.
(3) Find the average sum of squared errors of all points between the points A and B using:

$$E = \frac{1}{P_{i+1} - P_i} \sum_{n=P_i}^{P_{i+1}} (Y_n - m_i X_n + C_i)^2 \tag{15}$$

where P_i denotes the *index* of the i-th node from the waypoints list on the path list.
(4) If the average sum of squared error between two consecutive waypoints on the list is more than a threshold value, take the centre between the two points on the path list and add it to the waypoints list.
(5) Perform steps 3 and 4 recursively until all the average sum of squared error values are below the threshold value.

An example of the implementation above is shown in Fig. 10.

Fig. 10. Split-and-merge algorithm performed on the path generated by the A* algorithm.

4.4. *Global Convergence of Solution*

Given a map with at least one feasible path from a starting point $\mathbf{P_S}$ to a target point $\mathbf{P_T}$, the A* algorithm is guaranteed to terminate and is complete given at least one feasible path. When a majority of the world map M_G has not been explored, the local map M_L will only have obstacles wherever the UAV has previously observed obstacles in the global map. Let n_G and n_L denote the number of feasible paths from $\mathbf{P_S}$ to $\mathbf{P_T}$ in the M_G and M_L respectively via the sets N_G and N_L. Assuming that all UAV observations are true to the global map, it is hence given

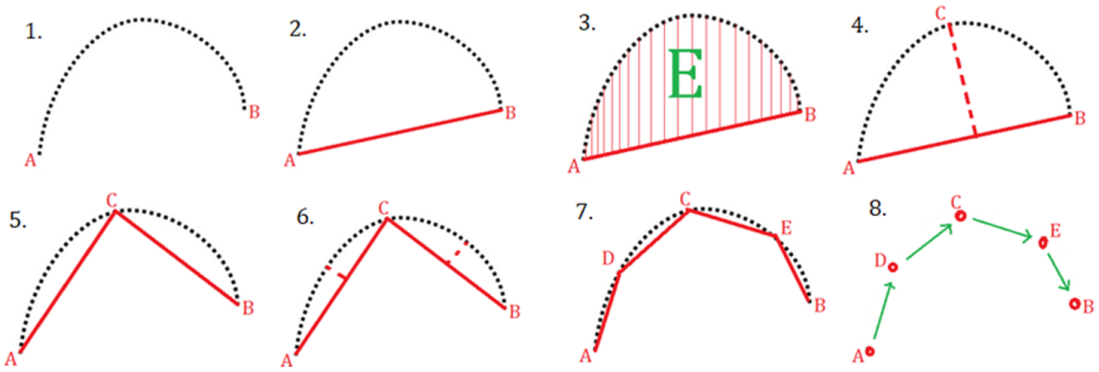

Fig. 9. Split-and-merge algorithm.

that $|M_L|_0 \leq |M_G|_0$ where the $||_0$ operator denotes the L0 norm. To put succinctly, this simply means that the number of obstacles in the local map will always be less than or equal to the number of obstacles in the global map. By this notion, it is therefore easy to see that $n_L \geq n_G$, $N_G \in N_L$ and that $n_L \to n_G$ as $M_L \to M_G$.

Whenever the A* function is called, any one of the feasible paths $N_L^{(i)}$ in N_L will be used to reach $\mathbf{P_T}$. If this turns out to not be a feasible path in N_G, then the UAV will have to traverse to the unexplored regions in order to update M_L such that $N_L^{(i)}$ is removed from N_G. It is through this forced exploration that the UAV is pushed to have constant exploration in the world whenever it is not at $\mathbf{P_T}$. To this end, the A* algorithm is called whenever the UAV reaches a local minimum as detailed in Section 4.2.

The next step is to simply prove that this is the only possible outcome whenever a local minimum is reached. Doing this is simple, there are two sources to the UAV's movement - the vector field and the potential field. At any point in time, there are no vectorial components that point away from $\mathbf{P_T}$. From this factor alone, local minima do not exist as the UAV will always move toward $\mathbf{P_T}$. In fact, without the potential field component of the algorithm, the algorithm will push the UAV towards flying through walls, which would be disastrous. The core component preventing this scenario is the potential field. Potential fields have zero curl component. This implies that at no point in time can the UAV enter cyclical local minima - the only ones that exist are stationary points in the form of zero gradient points, at this point the UAV is stationary which would trigger the calling of the A* algorithm. The calling of the A* algorithm forms new intermediary waypoints for the UAV, which alter its flight path to take another one of the feasible paths in N_L.

One caveat to the above is when a waypoint and an obstacle occupy the same location, which would cause the UAV to continually fly circles around the obstacle in an attempt to reach the waypoint within. This however, is an easy fix, one only needs to ensure that the waypoint reached distance threshold, D_{TR} - the measure of minimum distance from waypoint before the UAV declares that the waypoint is reached - is more than the proximity allowance, D_{prox}. Doing this, the UAV will declare waypoint reached before it is allowed to enter this special case of cyclical local minima.

5. Results

Unifying each component of the algorithm yields the exemplary performance shown in Fig. 11 in Simulation. From here, a performance analysis of the OAN algorithm is done. At the time of writing, there have been multiple international efforts to provide meaningful benchmarks for various UAV OAN algorithms [34, 35, 36]. However, not one of them has been agreed upon to suitably benchmark UAV OAN algorithms. This is in large part due large variance in UAV design, from sensor input, actuator type, vehicle capability, and computational allowance of each UAV setup. Therefore, COAA* will instead be evaluated in several scenarios frequently described in literature.

This section starts by comparing the performance of COAA* in the real-world with those in simulation. This is to ensure that the simulated results track real-world results accurately enough.

After that, an ablation study of sorts will be done on COAA*. Given that COAA* is a highly scalable and modifiable algorithm, there exist a whole plethora of hyperparameters that can be tuned. This study will study which hyperparameters will provide the highest influence towards the algorithm performance.

At the end, the algorithm will be put through several interesting scenarios of repeatability in simulated real-world scenarios.

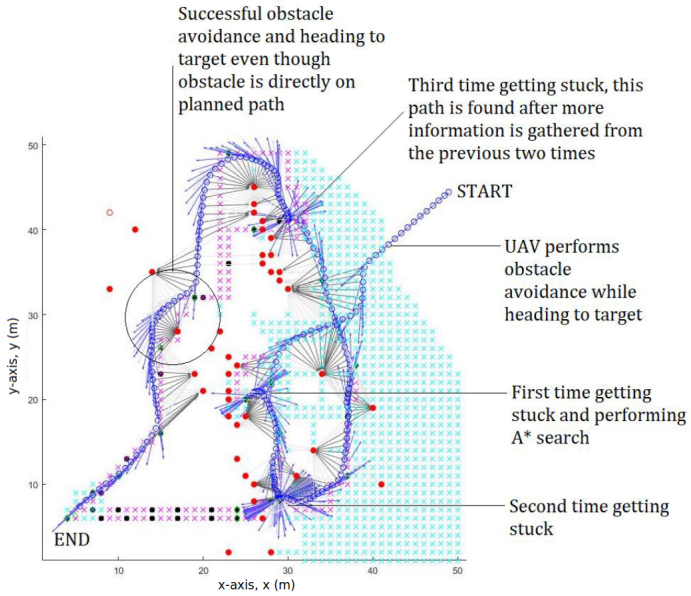

Fig. 11. Examplary performance of the algorithm demonstrating each component in simulation.

5.1. *Simulated Performance vs. Real-World Performance*

Two different scenario maps were constructed both in simulation and in real life, as shown in Fig. 14. The maps were designed to be easily constructed and not require the path finding implementation to be utilized since the A* algorithm is not subject to variations in dynamics between the real world and simulation. The chosen target was for the UAV to reach a waypoint located 15 meters away from the takeoff point. This small range of flight is largely due to a lack of allowable flight area at the time of writing. More in depth analysis of the algorithm in larger real world scenarios will have to be done at a later date.

The algorithm was tested with two sets of parameters, one set per map, as shown in Table 1.

The data of simulated flight path were then compared against the actual flight path of the UAV, along with the velocity data. Fig. 16 shows the actual airborne UAV in our experimentation in one of the maps. Note that traffic cones, instead of actual obstacles, are used to represent the locations of 1 meter wide obstacles as a safety precaution.

Table 1. Hyperparameters used on two separate maps to compare real world performance with simulated performance.

Hyperparameter	Test Set 1	Test Set 2
Proximity Allowance, D_{prox}	3 m	3 m
Maximum Velocity, $V_{\max}$	3 m/s	1.5 m/s
Acceleration Time Constant, K	0.5 s	0.2 s

The 2D trajectories of both maps are shown in Fig. 15.

The velocity profiles are shown in Fig. 12 and Fig. 13. In the second test case, the UAV stumbled around the 10 to 12 second mark, and this is likely due to controller tracking error under gusts of wind. In addition, the obstacles in the real world can only be replicated as accurately as they can be realistically placed. For this reason, a reasonable amount of tracking error is to be expected. On a whole, the simulated algorithm somewhat accurately replicates real world performance.

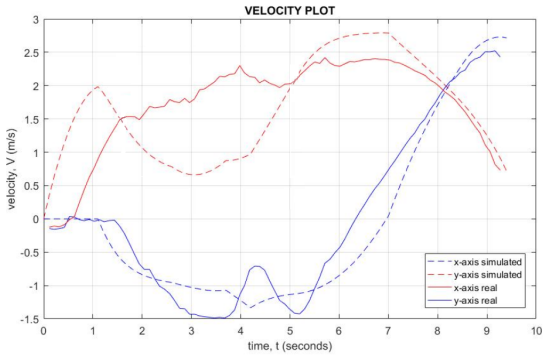

Fig. 12. Velocity profile for the first test set.

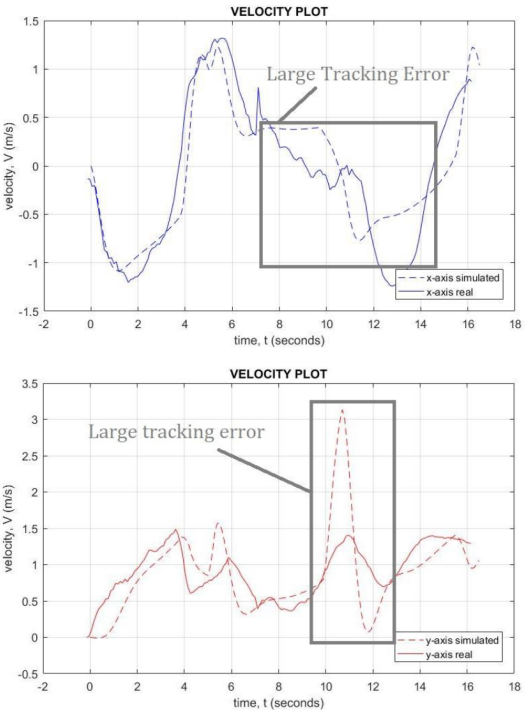

Fig. 13. Velocity profile for the second test set.

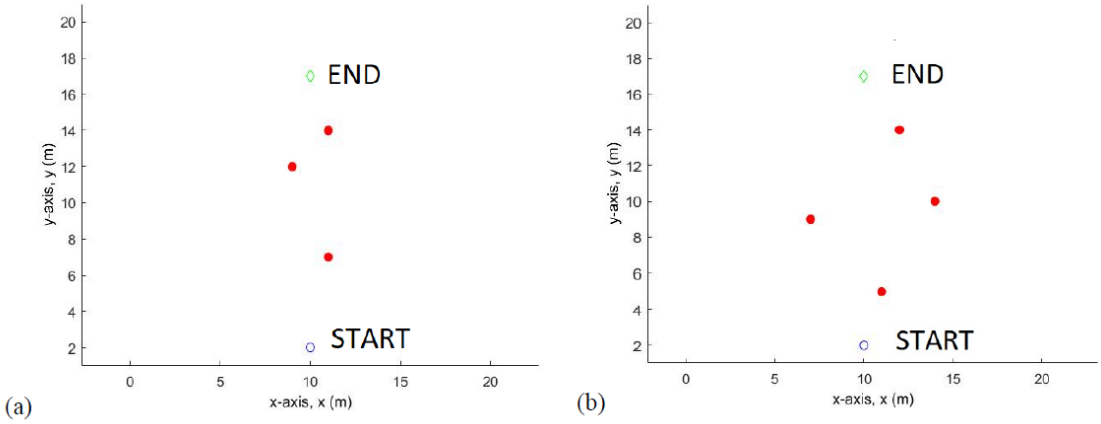

Fig. 14. Maps used to compare the UAV OAN algorithm in simulation and the real world. (a) Map 1. (b) Map 2.

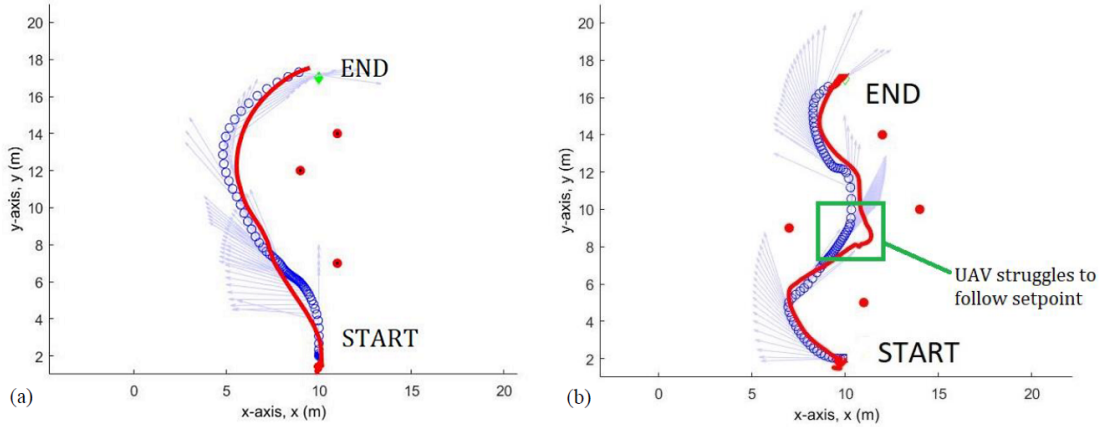

Fig. 15. 2D trajectory for the flight tests against the simulated results. The red line represents the real world UAV trajectory, while the blue circles represent the simulated UAV trajectory. (a) Map 1. (b) Map 2.

Fig. 16. Actual picture taken during the UAV test flights

5.2. *Hyperparameter Ablation Study*

Once the algorithm has been verified to work in the real world, a psuedo-ablation study on the algorithm hyperparameters was done. The main goal of this section is to analyze the influence of various hyperparameters on the algorithm performance. The evaluation procedure is as follows:

(1) List available hyperparameters to be tested.
(2) Set range of values to test hyperparameters.
(3) Identify performance metrics.
(4) Construct experimental maps to test hyperparameters.
(5) Test all available combinations of hyperparameters on every map.
(6) Gather data and performance metrics.
(7) Analyze gathered data with ANOVA and Pearson's Correlations Test.

The selected hyperparameters to be tested is shown in Table 2, while the evaluation metrics are shown in Table 3. The metrics are chosen such that lower is better for

Table 2. Hyperparameters used in ablation study and their respective range.

Hyperparameter	Range	Units	Description
Observation Radius, R_{OBS}	[5, 10, 15, 20]	m	Observation radius of the UAV.
Proximity Allowance, D_{prox}	[1, 3, 5]	m	Closest allowable distance between UAV and any obstacle.
Maximum Velocity, V_{max}	[3, 5, 7, 10]	m/s	Maximum allowable flight velocity.
Acceleration Time Constant, K	[0.2, 1.0, 2.0, 5.0]	s	Time constant for velocity profile smoothing
Waypoint Reached Threshold, D_{TR}	[1, 3, 5]	m	Distance between UAV and any waypoint to declare that the waypoint has been reached
Stuck Velocity Threshold, V_{GG}	[0.2, 0.5, 1]	m	Minimum velocity threshold before UAV is declared to be stuck at a local minimum.

Table 3. Hyperparameters used in ablation study and their respective range.

Performance Metric	Symbol	Units	Upper Range	Lower Range	Flight Characteristics Indicator
Mission Completion Time	T_C	s	$+\infty$	0	UAV speed
Maximum Proximity Allowance Breach	D_{PAB}	-	1	0	Safety
Number of A* Runs	n_{A*}	-	$+\infty$	0	Computational requirement
Maximum Jerk	J_{max}	m/s^{-3}	$+\infty$	0	Physical flight systems demand
Average Jerk	J_{ave}	m/s^{-3}	$+\infty$	0	Flight smoothness

each one. In addition, a did-not-finish (DNF) criteria was declared for simulations in which the UAV breaches the proximity allowance by more than 50%, or when the UAV fails to traverse the map after getting stuck in a location whereby its own hyperparameters deem it unable to complete the map. A set of maps created for this study are shown in Fig. 17, while the properties and description of each map are shown in Table 4.

The result of the ablation study is a 6-factor, 5-level analysis involving 1,728 independent interactions varied across four different maps.

The data gathered is statistical in nature. As a result, multiple statistical tools can be leveraged to analyze the data. Two were used in this study. The first is a 5-by-6-way Analysis of Variance (ANOVA), performed at each level of the data, on a per map basis. This tool permits the observation of the significance of each hyperparameter. The indicator of influence is known as a P-value derived from an F-value. A lower P-value denotes stronger evidence that the hyperparameter plays a significant role in influencing the performance of the algorithm. A statistically significant P-value of 0.005 is used in this study.

The second tool used is known as Pearson's Correlation Coefficient. This coefficient is a measure of the strength and type of association between two variables. A positive coefficient of greater magnitude denotes a stronger positive interaction between two variables; vice versa. Pearson's Correlation Coefficient has a range of -1 to 1.

Combined, these two tools allow the visualization of two things – the significance of influence of a hyperparameter and the type of influence of a hyperparameter. In summary, we found that the parameter that influences the performance of the algorithm most is the proximity allowance, D_{prox}. When set too high, like 5 meters, the algorithm fails to complete most maps, which is understandable as the algorithm is no longer allowed to traverse gaps smaller than 5 meters in width. The second and third highest influencing hyperparameters are the maximum velocity, V_{max} and acceleration time constant, K_1. Higher values generally yielded better performance, at the expense of higher demands on the UAV physical systems. A summary of Pearson's Correleation for the hyperparemeter influence on the algorithm performance is shown in Table. 5, in descending order of P-value from left to right.

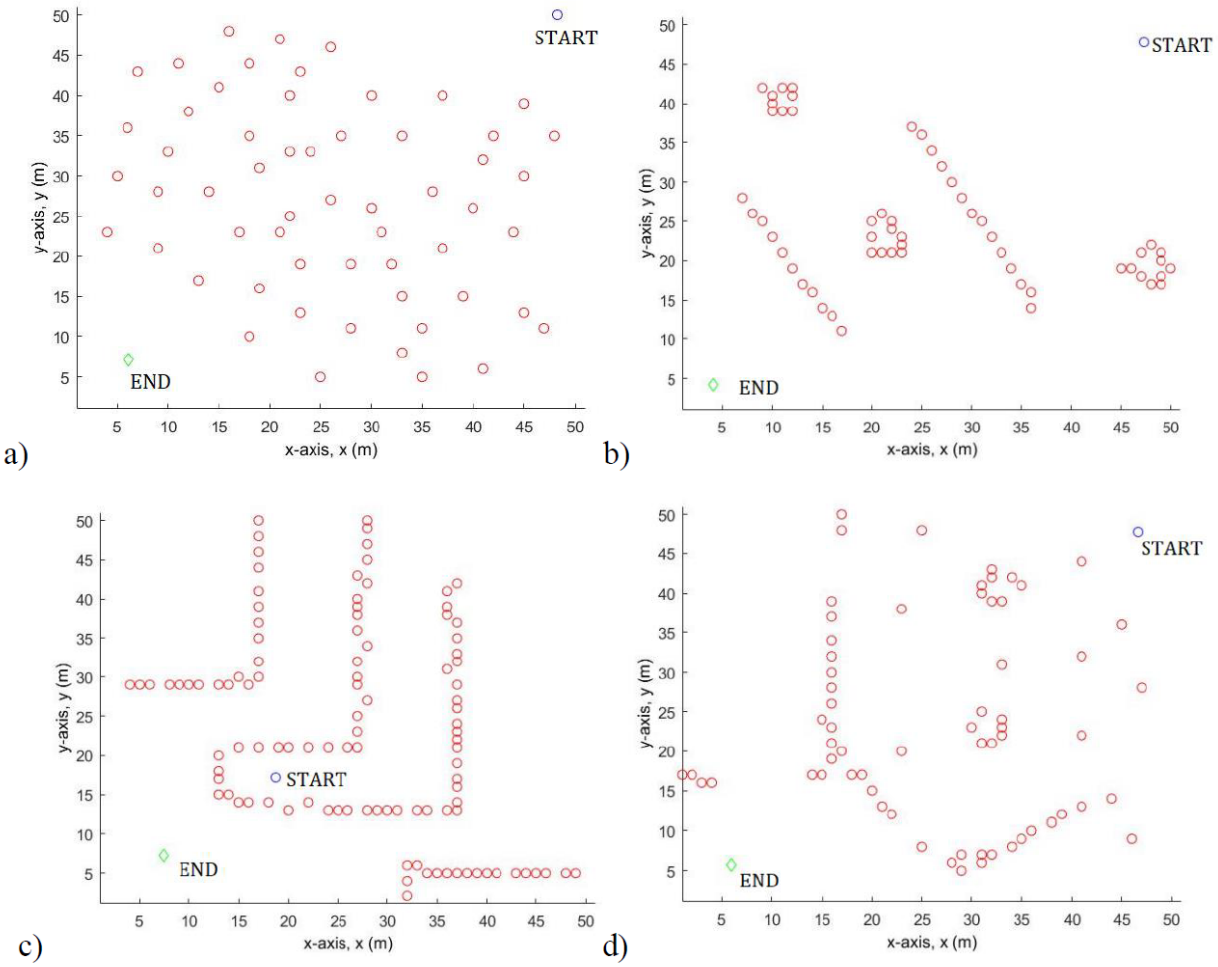

Fig. 17. (a) 'Sparse' map space. (b) 'Two walls' map space. (c) 'Corridor' map space. (d) 'Mixed' map space.

Table 4. Description and purpose list for the four maps used to evaluate hyperparameter performance and component interaction.

Map Name	Description	Simulated Space
Sparse	Randomly placed obstacles scattered throughout the map space.	Forest regions with randomly grown trees.
Two Wall	Almost parallel placed walls staggered from target location to UAV position. Scattered blocks throughout the map.	Urban Landscapes.
Corridor	Series of connected walls of semi-equal separation distance.	Indoor location made up of corridors.
Mixed	Mixture of all the different maps.	Mixture of everything.

Table 5. Pearson Correlation table of hyperparameter influence against algorithm performance.

	D_{prox}	V_{max}	K_1	R_{OBS}	D_{WP}	V_{GG}
T_C	0.7548	-0.4529	-0.0048	0.02300	-0.0303	-0.0340
D_{PAB}	0.1815	0.04213	0.05617	-0.0207	-0.0077	0.0071
n_{A*}	0.7325	-0.1515	-0.1091	-0.0637	-0.0197	0.0677
J_{max}	0.4405	0.4900	-0.4131	-0.0700	-0.0287	0.0060
J_{ave}	0.3334	0.4536	-0.2652	-0.1183	-0.0097	-0.0363

5.3. *Repeated Performance Analysis*

This section explores the ability of the algorithm to perform in a real world scenario with a properly configured set of hyperparameters, displayed in Table 6.

Table 6. Hyperparameters used to evaluate the algorithm in real world-esque simulations.

Hyperparameter	Symbol	Value
Observation Radius	R_{obs}	10 m
Proximity Allowance	D_{prox}	2 m
Maximum Velocity	V_{max}	5 m/s
Acceleration Time Constant	K_1	0.2
Waypoint Reached Threshold	D_{WP}	3 m
Stuck Velocity Threshold	V_{GG}	1 m/s

We use the floor plan of a real factory map [37], adapted for our purpose, shown in Fig. 18. The first run of the UAV is shown in Fig. 19. At the beginning, the performance of the algorithm behaves suboptimally as it is largely unaware of its surroundings. A lot of wall searching is done.

When the algorithm is made to preform the run again, the UAV is able to immediately take the most optimum path, as a consequence of having already knowing the map. Heading to the target is now simply a case of executing a single A* pass and then following the waypoints generated. This is shown in Fig. 20. Interestingly, the algorithm opts to pick the more optimum path, prioritizing speed and safety as opposed to the first explatory run.

When the map experiences a modification, as in the case when we close off one of the passage ways the UAV is allowed to take, exploration once again happens in the area where the modification is made. This is the case of the algorithm updating its internal map with new knowledge, even in a partially observable environment. This example is shown in Fig. 21. In the case when the map stays the same and the start and end points are changed, the algorithm, behaves well as in the case of Fig. 22.

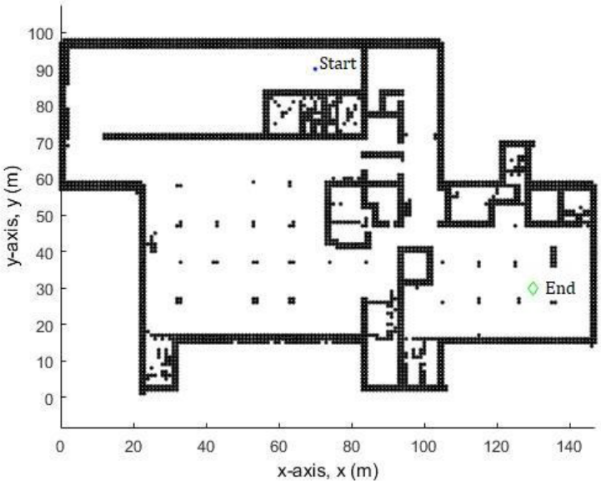

Fig. 18. Factory map adapted for our purpose.

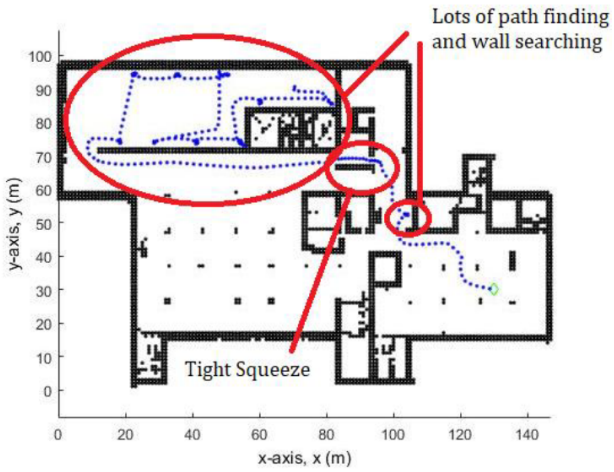

Fig. 19. First run of the UAV in the factory map. A lot of exploration occurs as the UAV tries to learn about its surroundings.

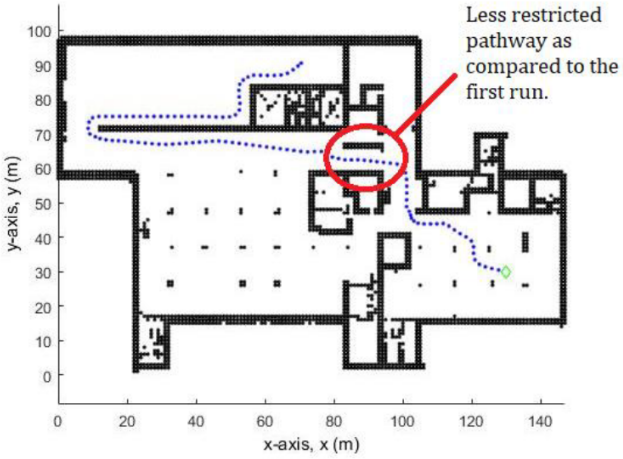

Fig. 20. Second run of the UAV in the factory map. No more exploration is needed as the algorithm figures out the most optimum path right from the get go.

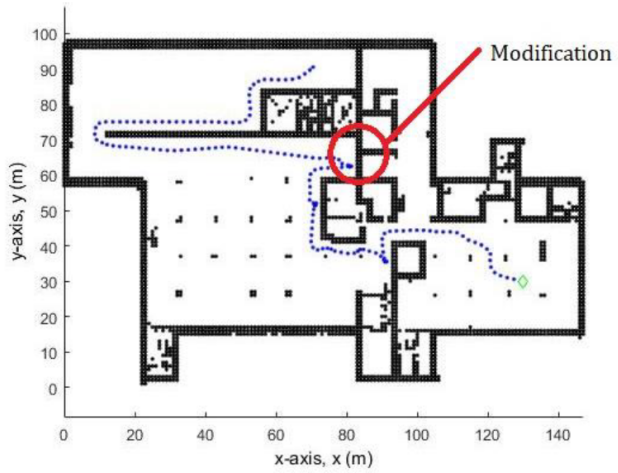

Fig. 21. When the map is changed, the algorithm is able to suitably adapt to find another appropriate path to reach the target.

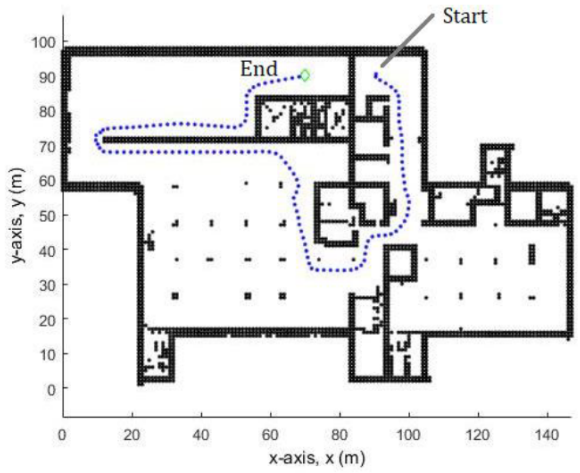

Fig. 22. When the start and end points are changed, smooth performance is the direct result of the algorithm having a more informed local map.

6. Conclusion

In this work, we proposed COAA* - an obstacle avoidance and navigational algorithm for unmanned aerial vehicles designed from first principles methods. The design focuses on a few things, namely, computational efficiency, scalablility, and versatility. The algorithm emphasises the delayed usage of computationally expensive operations, namely the A* algorithm. The scalability is derived from the fact that every component of the algorithm can be easily swapped out with another algorithm without much consequence. In addition, the algorithm is able to cater to a wide variety of sensor suites as the algorithm simply assumes an occupancy grid. The obstacle avoidance component of the algorithm can easily be adapted to be built upon other algorithms such as object tracking. We have tested the algorithm on an Intel UP Squared single board computer to validate its performance, and have then extensively tested the algorithm in a variety of maps with different hyperparemeters in computer simulation.

References

[1] H. Lee and H. J. Kim, Trajectory tracking control of multirotors from modelling to experiments: A survey, *International Journal of Control, Automation and Systems* **15**(1) (2017) 281–292.

[2] M. Achtelik, A. Bachrach, R. He, S. Prentice and N. Roy, Stereo vision and laser odometry for autonomous helicopters in gps-denied indoor environments, *SPIE Unmanned Systems Technology XI*, **7332** (2009).

[3] D. Cheng, A. C. Charles, S. Srigrarom and H. Hesse, Morphing concept for multirotor uavs enabling stability augmentation and multiple-parcel delivery, *AIAA Scitech 2019 Forum*, (2019), p. 1063.

[4] S. M. Adams and C. J. Friedland, A survey of unmanned aerial vehicle (uav) usage for imagery collection in disaster research and management, *9th International Workshop on Remote Sensing for Disaster Response*, **8** (2011).

[5] I. Mademlis, V. Mygdalis, N. Nikolaidis and I. Pitas, Challenges in autonomous uav cinematography: An overview, *2018 IEEE International Conference on Multimedia and Expo (ICME)*, IEEE (2018), pp. 1–6.

[6] T. Adão, J. Hruška, L. Pádua, J. Bessa, E. Peres, R. Morais and J. Sousa, Hyperspectral imaging: A review on uav-based sensors, data processing and applications for agriculture and forestry, *Remote Sensing* **9**(11) (2017) p. 1110.

[7] J. Cho, G. Lim, T. Biobaku, S. Kim and H. Parsaei, Safety and security management with unmanned aerial vehicle (uav) in oil and gas industry, *Procedia Manufacturing* **3** (2015) 1343–1349.

[8] S. Nunna, A. Kousaridas, M. Ibrahim, M. Dillinger, C. Thuemmler, H. Feussner and A. Schneider, Enabling real-time context-aware collaboration through 5g and mobile edge computing, *2015 12th International Conference on Information Technology-New Generations*, IEEE (2015), pp. 601–605.

[9] K. Lee, J. J. Tai and S. K. Phang, Bobby2: Buffer based robust high-speed object tracking, *arXiv preprint arXiv:1910.08263* (2019).

[10] 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, *2018 Fourth International Conference on Advances in Computing, Communication & Automation (ICACCA)*, IEEE (2018), pp. 1–6.

[11] 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* **30** (2015) 65–75.
[12] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, *Autonomous robot vehicles*, (Springer, 1986), pp. 396–404.
[13] 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* **54**(4) (2006) 288–299.
[14] D. Fu-guang, J. Peng, B. Xin-qian and W. Hong-Jian, Auv local path planning based on virtual potential field, *IEEE International Conference Mechatronics and Automation, 2005*, **4**, IEEE (2005), pp. 1711–1716.
[15] T. T. Mac, C. Copot, A. Hernandez and R. De Keyser, Improved potential field method for unknown obstacle avoidance using uav in indoor environment, *2016 IEEE 14th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, IEEE (2016), pp. 345–350.
[16] 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* **55** (2016) 174–184.
[17] D. Panagou, Motion planning and collision avoidance using navigation vector fields, *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE (2014), pp. 2513–2518.
[18] M. Radmanesh, M. Kumar, P. H. Guentert and M. Sarim, Overview of path-planning and obstacle avoidance algorithms for uavs: a comparative study, *Unmanned systems* **6**(02) (2018) 95–118.
[19] S.-Y. Shin, Y.-W. Kang and Y.-G. Kim, Reward-driven u-net training for obstacle avoidance drone, *Expert Systems with Applications* **143** (2020) p. 113064.
[20] A. Singla, S. Padakandla and S. Bhatnagar, Memory-based deep reinforcement learning for obstacle avoidance in uav with limited environment knowledge, *IEEE Transactions on Intelligent Transportation Systems* (2019).
[21] O. Walker, F. Vanegas, F. Gonzalez and S. Koenig, A deep reinforcement learning framework for uav navigation in indoor environments, *2019 IEEE Aerospace Conference*, IEEE (2019), pp. 1–14.
[22] H. Virani, D. Liu and D. Vincenzi, The effects of rewards on autonomous unmanned aerial vehicle (uav) operations using reinforcement learning, *Unmanned Systems* **9**(04) (2021) 349–360.
[23] W. Zhang, Y. Zhang and N. Liu, Danger-aware adaptive composition of drl agents for self-navigation, *Unmanned Systems* **9**(01) (2021) 1–9.
[24] J. Li, M. Ran, H. Wang and L. Xie, A behavior-based mobile robot navigation method with deep reinforcement learning, *Unmanned Systems* **9**(03) (2020) 201–209.
[25] Z. Y. Ng and S. K. Phang, Development of simultaneous localization and mapping algorithm using optical sensor for multi-rotor uav, *AIP Conference Proceedings*, **2233**(1), AIP Publishing LLC (2020), p. 030007.
[26] C. L. Hoo, S. M. Haris, E. C. Y. Chung and N. A. N. Mohamed, Steady-state integral proportional integral controller for pi motor speed controllers, *Journal of Power Electronics* **15**(1) (2015) 177–189.
[27] C. L. Hoo and S. M. Haris, Simulation of anti-windup pi controller, sipic on foc of pmsm, *ICIC Express Letters* **10**(11) (2016) 2539–2545.
[28] T. L. Chiah, C. Hoo, E. Chin and Y. Chung, Hardware simulation of a new anti-windup pi control for motor speed application, *J. Eng. Sci. Technol* (2017) 43–57.
[29] L. De Filippis, G. Guglieri and F. Quagliotti, A minimum risk approach for path planning of uavs, *Journal of Intelligent & Robotic Systems* **61**(1-4) (2011) 203–219.
[30] K. Peng, F. Lin, S. K. Phang and B. M. Chen, Nonlinear flight control design for maneuvering flight of quadrotors in high speed and large acceleration, *Unmanned Aircraft Systems (ICUAS), 2018 International Conference on*, IEEE (2018), pp. 212–221.
[31] S. K. Phang, K. Li, K. H. Yu, B. M. Chen and T. H. Lee, Systematic design and implementation of a micro unmanned quadrotor system, *Unmanned Systems* **2**(02) (2014) 121–141.
[32] A. Nemati and M. Kumar, Modeling and control of a single axis tilting quadcopter, *2014 American Control Conference*, IEEE (2014).
[33] Z. Benic, P. Piljek and D. Kotarski, Mathematical modelling of unmanned aerial vehicles with four rotors, *Interdisciplinary Description of Complex Systems*, **14**, INDEC (2016), pp. 88–100.
[34] S. Kasim, L. Y. Xia, N. Wahid, M. F. M. Fudzee, H. Mahdin, A. A. Ramli, S. Suparjoh and M. A. Salamat, Indoor navigation using a* algorithm, *International Conference on Soft Computing and Data Mining*, Springer (2016), pp. 598–607.
[35] P. Reist and R. Tedrake, Simulation-based lqr-trees with input and state constraints, *2010 IEEE International Conference on Robotics and Automation*, IEEE (2010), pp. 5504–5510.
[36] W. Nowak, A. Zakharov, S. Blumenthal and E. Prassler, Benchmarks for mobile manipulation and robust obstacle avoidance and navigation, *BRICs Deliverable D* **3** (2010) p. 1.
[37] P. Fletcher, Founding & requirements of starting a plastics product factory: Case hämeen lanka (2013).

Jun Jet Tai received his B.Eng. in Me-

chanical Engineering from Taylor's University, Malaysia, in 2020. He is currently pursuing his Ph.D. at Coventry University, United Kingdom. Primarily, his research interests include artificial intelligence and reinforcement learning. Secondarily, he is interested in state estimation and control for robotics, primarily of multi-rotor unmanned aerial systems (UAS).

Jun Jet was a research assistant in Taylor's University for 2 years during his undergraduate studies. He has since presented in several international conferences. He is the recipient of the Taylor's University Best Student Award; EURECA Conference Best High Impact Research Award 2019; team leader of the winning team during the AIRBUS Innovation Fun Day 2019 in Kuala Lumpur; and an invited participant to the CDIO Drone Academy 2018 in Kanazawa, Japan.

Swee King Phang received his B.Eng. and Ph.D. degrees in Electrical Engineering from the National University of Singapore, Singapore, in 2010 and 2014, respectively. He is currently a Lecturer with Taylor's University, Subang Jaya, Malaysia. His research interests include system and control of unmanned aerial systems (UAS), including modeling and flight controller design of the UAS, indoor navigation of the UAS, and trajectory optimization.

Dr. Phang has authored numerous publications, including a book chapter in the Handbook of Unmanned Aerial Vehicles, multiple peer-reviewed journals and International conference proceedings. He is also the recipient of the Best Paper Award (Guan Zhao-Zhi Award) in the 33rd Chinese Control Conference, Nanjing, China (2013); finalist of the Best Paper Award in the 14th IEEE International Conference on Control and Automation, Alaska, US (2018); and IET Commendable Paper Award in 13th International Engineering Research Conference (2019). He is currently the Cluster Leader of Autonomous Robots Research Cluster at Taylor's University. Student teams from his research cluster have won numerous UAS related competitions, which include the champion for the AIRBUS Innovation Fun Day 2019 in Kuala Lumpur, Malaysia; and the champion for the CDIO Drone Academy Challenge 2018 in Kanazawa, Japan.

Felicia Yen Myan Wong received her B.Eng. in Mechanical Engineering from the University of Nottingham Malaysia Campus and her Ph.D. from the University of Nottingham Ningbo China in 2010 and 2017, respectively. She is currently a lecturer at Taylor's University, Subang Jaya, Malaysia teaching design thinking, material science and project management. A majority of her research interests, publications in journals and conference proceedings are about the development of environmentally friendly antifouling technology through surface modification of topographies to prevent biofouling growth using benchtop laboratory methods, 3D printing and Computational Fluid Dynamics. Additionally, she has also applied her expertise in numerical and statistical analysis to explore other research areas in the field of biomimicry and aerodynamics specifically in the design of morphing wings of aerofoils.