Depth-based Sampling and Steering Constraints for Memoryless Local Planners

Binh T. Nguyen1 · Linh Nguyen1 · Tanveer A. Choudhury1 · Kathleen Keogh1 · Manzur Murshed2

Received: 30 March 2023 / Accepted: 19 September 2023 / Published online: 21 October 2023
© The Author(s) 2023

Abstract
By utilizing only depth information, the paper introduces a novel two-stage planning approach that enhances computational efficiency and planning performances for memoryless local planners. First, a depth-based sampling technique is proposed to identify and eliminate a specific type of in-collision trajectories among sampled candidates. Specifically, all trajectories that have obscured endpoints are found through querying the depth values and will then be excluded from the sampled set, which can significantly reduce the computational workload required in collision checking. Subsequently, we apply a tailored local planning algorithm that employs a direction cost function and a depth-based steering mechanism to prevent the robot from being trapped in local minima. Our planning algorithm is theoretically proven to be complete in convex obstacle scenarios. To validate the effectiveness of our DEPth-based both Sampling and Steering (DESS) approaches, we conducted experiments in simulated environments where a quadrotor flew through cluttered regions with multiple various-sized obstacles. The experimental results show that DESS significantly reduces computation time in local planning compared to the uniform sampling method, resulting in the planned trajectory with a lower minimized cost. More importantly, our success rates for navigation to different destinations in testing scenarios are improved considerably compared to the fixed-yawing approach.

Keywords Quadrotor · Depth image · Steering · Sampling · Local planning

1 Introduction
In recent years, depth perception has become available and more affordable, increasing its usage for robotic applications in the industrial and research communities [1, 2]. One interesting use case is on quadrotors, where onboard depth sensors generate data for obstacle avoidance algorithms (path planners) to keep the drone safe. Many research groups have employed depth data in their studies on path planning for quadrotors. However, those depth data remain underutilised in existing works, giving us an opportunity to further exploit them to increase sampling efficiency and path planning performance.

The sampling-based technique is ubiquitously used as a trajectory planning approach for quadrotors in unstructured environments [3]; however, the number of sampled collision-free trajectories is limited. A sampled set with a large percentage of in-collision candidates will require a significant amount of computational power on collision checking. For instance, Florence et al. [4] exploited their piecewise triple-double integrator in time for state modelling to produce a closed-form future maneuver library. Although their probabilistic maneuver library can be quickly propagated by uniformly sampling in the state space, it can also be rich in in-collision candidates. Since they utilise a k-d tree representation of local structures for the collision checking process, it requires up-front computational cost to construct that 3D structure. Also using k-d trees for collision checking, Lopez

1 Institute of Innovation, Science and Sustainability, Federation University, Northways Road, Churchill, Victoria 3842, Australia
2 School of Information Technology, Faculty of Science, Engineering and Built Environment, Deakin University, Burwood Highway, Burwood, Victoria 3125, Australia
et al. [5] generated a set of motion primitives specified by desired speeds and heading angles. They uniformly sample possible headings from within the field of view (FoV) while keeping the forward speed constant. Those motion primitives can easily satisfy state constraints, such as remaining within the FoV of the sensor. This approach may work effectively in environments with sparse obstacles. However, in an obstacle-dense area that would increase the binary search of a single k-d tree query, an unrefined motion primitive set can cause an exploded computation load for collision checking. Additionally, in [6], Ryll et al. uniformly sampled over yaw-heading and final Euclidean distance from the vehicle to generate a spatially and dynamically diverse set of motion primitives, which will also suffer from the same potential issue of heavy workload. While the works mentioned above fuse recent depth images into a local 3D structure, RAPPIDS - Rectangular Pyramid Partitioning using Integrated Depth Sensors [7] is more lightweight since it exploits only the latest depth image for directly partitioning the free space using a number of rectangular pyramids. However, RAPPIDS explores the space by randomly sampling trajectories’ endpoints over a given range in the sensor’s FoV (sampling space), which is not carefully constrained before sampling. This approach results in a low-quality sampled set that includes many in-collision trajectories when the quadrotor flies in an obstacle-filled area,burdening the collision checking.

Besides computational efficiency, planning performance can also be improved by directly utilising depth data. All those previously mentioned planners are classified as map-less and memoryless algorithms utilising depth cameras. Mapless planners [4–6, 8] only employ a local map that is just enough for local planning, and a memoryless one [7] plans directly on sensor data. By the use of monocular cameras, optical flow [9–11], and feature tracking [12–14] can be extracted to develop the memoryless path planning algorithms. On the other hand, a memoryless planner can also be learned given advancement in machine learning. For instance, Loquercio et al. [15] generated high-speed trajectories directly from depth images and current states by utilizing imitation learning. The authors of [16, 17] leveraged reinforcement learning to predict probabilities of events such as collisions, going over bumpy terrain, or human disengagement to choose the likeliest collision-free trajectories. Likewise, Nguyen et al. [18] employed a deep prediction network to account for uncertainty in state estimation while predicting collision probabilities. Nevertheless, due to model uncertainty of the learning-based planning approaches, the robot may still collide with obstacles or come to a dead end, which can ultimately degrade overall success rates. Distinguishing from those, map-based planning systems [19, 20] generally integrate global maps [21, 22] a priori and a global planning algorithm such as RRT* [23] to guarantee planning completeness in tasks such as exploration [24–28] or navigating toward a goal [29–31]. All mapless and memoryless planners cannot access any map but use sensor data. They are likely to fail to reach the goal in cluttered environments. For example, [32] introduces an autonomous system employing RAPPIDS to planning trajectories toward a goal in a cluttered environment. Their algorithm would stop planning when facing a large obstacle such as a wall. Therefore, there is still room for enhancement of the memoryless planner’s performance and ability to plan and find a solution autonomously.

In those mapless and memoryless algorithms, motion primitives are sampled without considering available depth information. We can utilise available depth data to refine the motion primitive set before it comes to collision checking. This is where our depth-based sampling method contributes. Specifically, we find and reject obscured primitives’ endpoints or constrain the sampling space only by querying depth values, automatically excluding many meaningless (in-collision) candidates in the sampled set. Since this procedure can be done in constant time, our approach will reduce the computational workload required in collision checking. Additionally, we will promote the planning autonomy level by proposing a steering mechanism, increasing the system’s success rate. Our steering mechanism uses a heuristic algorithm coupling with a direction utility function to offer existing memoryless planners a higher autonomy in scenarios of convex obstacles.

To evaluate our proposed approaches, we build a complete planning system that is capable of autonomously navigating a quadrotor through a cluttered environment to a given destination. The planning system consists of only a memoryless local planner, and no explicit global planners are required. Results show that our depth-based steering mechanism can navigate through large-obstacle scenarios in which the existing depth-based local planners will get to a dead end. That is, our approach enables successful autonomous flights toward a goal without the integration of high-level global planners. Another simulation benchmark also demonstrates that our depth-based sampling method will reach a considerably higher computational efficiency than the original uniform sampling technique used in previous works [7]. To the best of our knowledge, this is the first work that integrates a depth-based sampling technique and an autonomous steering mechanism within a memoryless planner framework for navigating toward a goal in unknown cluttered environments.

Our contributions can be summarized as follows:

1) We propose a novel depth-based sampling technique for memoryless planners to improve their computational efficiency using depth sensors. Our sampling mechanism is able to generate better trajectories than RAPPIDS [7] inside a computing time interval allocated to a planning sequence.
2) We propose a new local planning algorithm that is theoretically proven to be complete in convex obstacle scenarios. Our algorithm was validated in simulations, where the obtained results demonstrate its outperformance as compared with the previous memoryless planner [32] in terms of escaping large (larger than the FoV) convex obstacles.

2 Memoryless Local Planning Problem

Mapless and memoryless planners only use a very limited dynamic memory for collision checking. Mapless planners [4–6, 8] fuse the most recent historical sensor measurements into a local 3D data structure (e.g., k-d tree) but not an entire global map for collision checking. A memoryless planner [7] exploits only the latest sensor data. These types of planners lend themselves to local planning purposes.

Figure 1 describes the workflow of a memoryless planner using depth sensors, where the positions of collision checking and cost evaluating steps are interchangeable. Each step would use different computational resources, depending on the types of planners.

For example, memoryless or mapless planners spend most of their resources on collision checking, while map-based planners may cost substantial computational power in both evaluating costs (e.g., volumetric exploration gain [33]) and collision checking (updating and querying a map). In both cases, motion primitive sampling is always the first step in the flow. Thus, its efficiency affects the performance of the entire workflow. The uniform sampling technique [4–8] only offers an unrefined sample set, which might contain a number of meaningless (in-collision) candidates. Such a low-quality sampled set (low in collision-free samples) will burden the collision-checking stage, thus it can be a bottleneck to the system’s performance. Checking for collision is the most computationally expensive task in the pipeline as we have to manipulate memory-consuming data structures such as depth images, k-d trees or Euclidean signed distance field (ESDF) [22]. If the sampling space contains too many in-collision candidates, planners would cost much more computational power and time in evaluating many trajectories that would result in a collision. Therefore, refining the sampling space can be a promising technique.

To lower the requirement for computing resources, we need to eliminate a certain number of in-collision candidates in the sample set by considering depth data. We can determine a particular type of in-collision trajectories by comparing the trajectory endpoint’s position with its projection in the depth matrix. If the endpoint is farther from the robot than its projection’s depth, it is considered inside obstacles, and its trajectory is considered in-collision, and vice versa. By analysing the relative position between trajectories and the occupied space, we categorize them into four types: (type (0)) collision-free trajectories have their entire path outside the occupied space, (type (1)) trajectories have their endpoints too close to the occupied space (the robot will collide when getting to the endpoint), (type (2)) trajectories have their middle sections inside the occupied space and their endpoints outside (the robot will collide before getting to the endpoint) and (type (3)) trajectories have their endpoints inside the occupied space as in seen Fig. 2.

Although the collision-checking process can reject all types of in-collision trajectories, the fewer in-collision candidates that need to be checked, the fewer computing resources will be spent. Therefore, in this work, we will systematically exclude the type (3), obscured-endpoint candidates, by constraining the sampling space using raw depth data. Type (1) and type (2) will be rejected by the collision-checking step afterwards. Reducing the number of in-collision candidates that require checking will reduce needed computing resources.

3 Depth-based Sampling Constraint for Local Planning

In this section, we will first present how depth-based constraints can be applied in two sampling approaches: in the
input space and in the state space. Next, a local planning algorithm to prevent the robot from being trapped in a dead end, exploiting a direction cost function and a depth-based steering mechanism. We later prove that the algorithm guarantees planning completeness (or global convergence to the target [34]).

3.1 Constrained Sampling Space

For local planning, motion primitives can generally be sampled in input space or state space. When sampling in input space, a dynamic model will be utilized to propagate a trajectory, e.g., constant-acceleration point-mass [4]. These propagated trajectories easily satisfy input constraints as they are only sampled in the valid range of the input space. However, uniform sampling in input space is not necessarily uniform in configuration space, where collision checking will be done. On the other hand, the sampling in state space can guarantee probabilistic optimality, but checking for dynamic feasibility has to be done afterwards [35]. Both sampling approaches (using input space and state space) do not consider depth perception before collision checking. The denser the environment is, the more in-collision trajectories are in the sampled set, wasting much computational resource in checking many in-collision candidates. Hence, we propose to utilize available depth data to preprocess sampling space or sampled set for both mentioned approaches (using input space and state space) in different pipelines shown in Fig. 3.

In the framework of this project, we do not consider the depth perception error. That is, the depth pixel’s values of the input frame are assumed to be perfectly accurate.

For sampling in input space as outlined in Fig. 3a, propagated trajectories’ endpoints will then be projected into the latest image plane. We consider a coordinate system with the axis $X_O$ and $Z_O$ parallel to the ground and $Y_O$ pointing downwards such that the quadrotor is at $[0,0,0]$ and the $XY$ plane is parallel to the image plane. The axis $Z_O$ is perpendicular to the image plane; thus, it points the depth direction. Note that we always have available depth matrix outputs directly from the depth sensor $D_{w \times h}(\mathbb{R})$ where $w$, $h$ are the width and the height (in pixels) of the input depth image. We also have that $(i, j), \forall i \in [0, w], j \in [0, h]$ are pixel coordinates of the endpoint’s projection in the image plane and the associated depth value $d_{i,j}$ by querying the depth matrix, $d_o$ and $d_p$ is the distance from the sampled endpoint to the robot in the $Z_O$ axis of the local frame before and after constraining, respectively. We then compare $d_{i,j}$ with $d_o$: if $d_o > d_{i,j}$, the current endpoint is obscured, and the corresponding trajectory is discarded; otherwise, we accept the sampled endpoint and the corresponding trajectory.

For sampling in state space as represented in Fig. 3b, we propose a technique for excluding these type (3) trajectories as in Fig. 2. Firstly, we decouple sampling the endpoints’ position into depth direction and $XY$ plane (image plane). We

---

![Fig. 3](https://via.placeholder.com/150)

**Fig. 3** Pipelines for applying depth-based constraints to two sampling approaches (in the input space and in the state space). Red dashed arrows are rejected trajectories whose endpoints are obscured. Green dash arrows are valid sampled candidates. While in the input space (a), depth constraints are used to discard invalid samples, they are used to confine sample space before sampling in the state space (b).
then adapt a constraint process to the depth sampling space (sample region). Specifically, we shrink the depth sample region (for each sampled pixel) to fit into an unoccupied region, pulling all the sampled endpoints into the free space. We directly utilize available raw data from a depth sensor to determine the unoccupied region for each sampled pixel \((i, j)\).

In sensor-based collision checking, we only sample in a reliable sensor range. Therefore, a predefined upper bound \(u\) and lower bound \(l\) will limit the sample region to the sensor’s reliable range \([l, u]\). Within this work, we choose \(a = 3\ m\) and \(l = 1\ m\), suitable for most popular depth sensors on the market, such as Intel® RealSense™ D435. If the associated depth value of the sampled pixel \(d_{i,j}\) falls inside the reliable range \([l, u]\) \((d_{i,j} \in [l, u])\), we scale down the initial depth sample region \([l, u]\) to the constrained depth sample region \([l, d_{i,j}]\) for each sampled pixel \((i, j)\). This manipulation will convert obscured sampled points into depth-based constrained sampled points, lying in the free space, as illustrated in Fig. 4.

If the associated depth value exceeds the upper bound \((d_{i,j} > u)\), in-range sampled points will always lie in the free space. If the associated depth is smaller than the lower bound \((d_{i,j} < l)\), the generated trajectory will collide with obstacles. However, in this case, we keep the in-range sampled point and let the collision-checking process rejects this in-collision candidate. We do not need to constrain the depth sample region in these two cases: \(d_{p} = d_{o}\). That means we are not directly rejecting any candidate in sampling but constraining the sampling space, maintaining its continuity and probabilistic optimality. The sampling space for trajectories’ endpoints before and after constraining is shown:

\[
\Omega_o = \{(i, j, d_o) | i \in [0, w], j \in [0, h], d_o \in [l, u]\};
\]

\[
\Omega_p = \{(i, j, d_p) | i \in [0, w], j \in [0, h], d_p \in [l, u] \setminus [d_{i,j}, u]\};
\]

where \(\Omega_o\), \(\Omega_p\), \((i, j, d_o)\), \((i, j, d_p)\) are original space, constrained space, and their corresponding sample points, respectively.

\[
d_p = \begin{cases} 
(d_o - l) \frac{d_{i,j} - l}{u - l} + l, & \text{if } d_{i,j} \in [l, u] \\
 d_o, & \text{otherwise}
\end{cases}
\]

It is noted that \((i, j, d_p)\) is just a sampled pixel and the corresponding depth value in the depth frame. We need to derive the actual endpoint \(P\) in the Cartesian space of the camera-fixed frame, as illustrated in Fig. 4, for trajectory generation. Therefore, once we have a valid pixel candidate after a given sampling cycle, the pixel \((i, j, d_p)\) will be deprojected to \(P\) by equation:

\[
P = \left(\frac{(i - c_x) \ast d_p}{f_x}, \frac{(j - c_y) \ast d_p}{f_y}, d_p\right)
\]

where \(f_x\), \(f_y\), \(c_x\), and \(c_y\) are the focal lengths and principle point’s coordinates in the corresponding axis of the camera, respectively. Details of the depth-based sampling method are described in Algorithm 1.

**Algorithm 1** Depth-based sampling.

**Input:** Latest depth image \(D\), user-defined range: lower bound \(l\), upper bound \(u\)

**Output:** Sampled point \(P\) as the endpoint’s position of the next trajectory for evaluating

1: function DEPTH_BASED_SAMPLE()
2:     Randomly sample a pixel \((i, j)\) from image frame;
3:     Randomly sample \(d_o \in [l, u]\);
4:     Query \(d_{i,j}\) value in \(D\);
5:     if \(d_{i,j} \in [l, u]\) then
6:         \(d_p = (d_o - l) \frac{d_{i,j} - l}{u - l} + l\);
7:     else
8:         \(d_p = d_o\);
9:     end if
10:    Deproject current candidate \((i, j, d_p)\) to a real-world endpoint \(P\);
11: end function

3.2 Planning Algorithm

We propose the trajectory planning pipeline as described in Fig. 5.

The local planner aims to find the lowest cost (with respect to an objective function) and collision-free trajectory over a given user-defined computation time. The planner will run many sequences until the time limit is reached. For each sequence, we generate a minimum-average-jerk polyno-
mial trajectory [35] using our depth-based sampling method described in Subsection 3.1. Our candidate trajectories are generated with constraints of zero terminal velocity and acceleration to drive the quadrotor to a hover state at the end of trajectories. Thus, if the local planner fails to generate new feasible trajectories, the last feasible candidate can be completely executed while the robot is always inside the free space boundary. The collision checking function follows the algorithm RAPPIDS [7]. This is the most computationally demanding step and is only called when the sampled trajectory’s cost is lower than the current best cost. Hence, the best trajectory’s cost converges to the lowest over time.

We propose a direction function $J(P)$ that minimize the direction cost of trajectories as follows:

$$O = (0, 0, 0)$$

$$\vec{E} = \frac{\vec{OG}}{\|\vec{OG}\|}$$

$$J(P) = \frac{\vec{E} \cdot \vec{PO}}{\|\vec{PO}\|}$$

where $O$, $G$, and $P$ are the camera-fixed frame positions of the vehicle, the goal and the trajectory’s endpoint, respectively; we already have $P$ from Eq. 3. $T$ is the execution time of the trajectory. $\vec{E}$ is the vector of exploration direction. This vector-based function corrects the velocity vector to the desired direction. Previous works [7, 32] employ a function prioritising the average velocity to the goal, allowing the lateral motion to be free. Thus it causes a more diverged and fluctuated path, even in completely free space. In our proposed system, $\vec{E}$ will be specified as the unit vector from the vehicle’s current position to the goal. The cost is simply the dot product of the exploration vector $\vec{E}$ and the negative unit vector of the motion primitive. That is, the direction of a better trajectory will align closer to the goal direction.

This feature also paves a platform for a combination with the autonomous steering mechanism to help the robot escape stuck motions.

### 3.3 Trajectory Generation for Stuck Motions

In this subsection, we present a new steering command for the local planner to generate feasible trajectories for the robot in stuck motions autonomously. A stuck motion happens when the vehicle faces a very close obstacle, and no feasible trajectory can be generated. Therefore, the robot rests at the last executed trajectory’s endpoint, called a dead end. This situation is popular in real-world scenarios. Any obstacle, such as a large wall of a building in a downtown or a truck on a road, which is larger than the camera’s FoV, would block the vehicle from moving. In ordinary operations, a normal trajectory can be generated by following the procedure of the pipeline described in Subsection 3.2. However, when the local planner cannot find any feasible conventional trajectory for one second, e.g., the vehicle is in stuck motions, steering commands are requested. The steering command is a motion primitive with the position as the latest executed trajectory’s endpoint and an additional yaw orientation. New depth data will come due to changes in the FoV as the quadrotor executes steering commands. The planner will consistently send steering commands until it finds at least one feasible trajectory from this latest new depth frame.

We propose a depth-based mechanism for generating these steering commands when the vehicle is at a dead end. We scan the latest depth image to locate the nearest point in the space, which means the closest obstacle. We divide space in half by a vertical plane that passes through the vertical centreline of the image. On which half-space the nearest point lies, the quadrotor steers toward the other half as illustrated in Fig. 6.

The desired yaw angle will be accumulated by steering value over time on the controller side and calculated as follows.

$$\gamma = \begin{cases} 
1, & \text{if } x_c < (w/2) \\
-1, & \text{otherwise}
\end{cases}$$

$$\psi_r = \psi_f + \gamma \ast K_p \ast \Delta t,$$

where
\( \psi \) is the sign value representing steering direction, \( \gamma = 1 \) means steering right, \( \gamma = -1 \) means steering left.

- \( x_c \) is the pixel location of the closest point along the x-axis of the image frame.
- \( w \) is the width of the image frame in pixel unit.
- \( \psi_r \) is the desired reference yaw angle for yaw control when executing a steering command.
- \( \psi_f \) is the feedback of the current yaw angle.
- \( K_p \) is the accumulation gain for the yaw angle.
- \( \Delta t \) is the cycle time of a control loop.

This trajectory generation scheme for each planning sequence is summarised in Algorithm 2.

Algorithm 2 Trajectory command generation pipeline.

1: while not at goal do
2:   generate normal best direction trajectory;
3:   if cannot generate any feasible normal trajectory for more than one second then
4:     execute steering commands by feeding \( \psi_f \) computed by Eq. 6, to the yaw controller;
5:   else
6:     send the best-direction trajectory found to the quadrotor;
7:   end if
8: end while

The depth-based steering is simple yet effective with the type of geometry-based planners since it is hungry for new input. When a sufficiently clear space view comes, the planner will generate feasible trajectories again, pulling the quadrotor out of infeasible configurations, especially when facing a massive obstacle that is even bigger than the FoV. The mechanism suits itself with direction-prioritized trajectory generation since it gradually provides a new angle of FoV. The robot will stop steering immediately once it finds a feasible trajectory. Without steering, yaw control is fixed. When no feasible trajectories are found in the current FoV, the robot will not escape the dead end if the FoV cannot be changed. We will later prove that our proposed planning algorithm is complete in convex obstacle scenarios and evaluate the success rate against the fixed-yawing policy.

3.4 Yaw Control in Obstacle-dense Areas

When it comes to travelling long distances over many obstacles, we need another mechanism to integrate the best direction trajectory and depth-based steering to control. Instead of applying a fixed yaw control that faces the camera toward the goal [32], we adopt a yaw control scheme that faces the camera to the local goal (the latest feasible trajectory’s endpoint) by setting the desired control yaw angle \( \psi_d \) equal to the bearing of the vehicle-goal vector \( \psi_b \). We only do that when the position error (distance to the goal) vector’s length \( \| r_{err} \| \) is greater than one meter. Otherwise, we set \( \psi_d \) equal to either \( \psi_r \) when the robot needs to do a steering command or the latest executed control yaw value \( \widetilde{\psi}_d \) when it does not (Algorithm 3).

Algorithm 3 Yaw control.

1: if \( \| r_{err} \| > 1m \) then
2:   \( \psi_d = \psi_b \)
3: else
4:   if is steering then
5:     \( \psi_d = \psi_r \)
6:   else
7:     \( \psi_d = \widetilde{\psi}_d \)
8:   end if
9: end if
10: \( \psi_d = \psi_d \)

It is noted that \( \psi_r \) is calculated in each planning cycle by Eq. 6. The function proportionally accumulates the yaw reference by the steer direction \( \gamma \) over time, which causes the vehicle to consistently steer in one direction until the planner tells it to do something else (e.g., track a new trajectory as described in Algorithm 2). That is, the system automatically prioritizes the exploration direction that navigates toward the free space, steering away from an obstacle before bumping into it. As a result, the quadrotor would autonomously find a clearer path, escaping cluttered areas faster and safer and avoiding more potential “stuck” situations (which need to steer).

3.5 Completeness

Here we formally prove the completeness of our proposed local planning algorithm described above. This work requires several reasonable assumptions to make the algorithm theoretically provable. First, we only consider convex obstacles, and

Assumption 1 It is assumed that the robot faces only one convex obstacle at a given time instance in a multiple-obstacle scenario.

Secondly, we assume there is no uncertainty in the control process, which means the robot can track the given path flawlessly. Finally, the planning algorithm is assumed to be locally optimal. Hence,

Assumption 2 The planner always finds the collision-free and minimal-cost trajectory according to the direction objective function in Eq. 4.

Considering scenarios of a single obstacle lying between the robot and the goal, let \( q_G \) be the predefined fixed location of the goal, the \( m \)-line be the instant line that passes through the robot’s current and \( q_G \) at a given planning cycle, steering point \( q_S \) is the closest intersection of the \( m \)-line and the
obstacle boundary at the starting time; leaving point \( q_L \) is the point where the \( m \)-line intersects the obstacle boundary at only one point. In a single obstacle scenario, the goal will be visible from \( q_L \) due to the convexity properties, as illustrated in Fig. 7a. The robot may need to execute a steering command (we now call the act of executing a steering command a "steer" for brevity) only when obstacles lie in the \( m \)-line.

**Lemma 1** When the vehicle faces a convex obstacle, it will reach a leaving point \( q_L \) after only the first steer.

**Proof** We refer readers to the Appendix.

**Theorem 1** The proposed local planning algorithm in Section 3 is complete in multiple convex obstacle scenarios with subject to Assumption 1.

**Proof** We refer readers to the Appendix.

### 4 Experiments and Results

This section will discuss experiments and related benchmarks for evaluating our proposed planner’s computational efficiency and planning performance compared to others.

#### 4.1 Algorithm Evaluation

We compared the planner’s performance regarding the cost of the best candidate found. We applied multiple Monte Carlo simulations\(^1\) [7] on the two policies: uniform sampling and depth-based sampling. The simulations were run offline on 1000 depth images of obstacle-dense scenarios. Depth images of cluttered scenes are generated by putting three sticks with different angles at random distances between 1 m and 2.5 m. Sticks are sized from 10 cm to 30 cm in radius. Three samples of generated scenarios are visualized in Fig. 8. Since the input for each planning cycle is only one depth image, two policies will plan on an identical image and the same sampling range for each cycle. It should be noted that the time interval for each cycle is given, as proposed in the planning algorithm described in Subsection 3.2. Therefore, each depth scene will be planned in different allocated time interval values ranging from 0.1 to 20 milliseconds. The average results of 1000 scenes for the two policies in each time interval value will then be summarized regarding the best-planned trajectory’s cost value, the number of trajectories evaluated and the number of collision-free trajectories found.

We repeated the same simulation setup on two hardware platforms: an i7-1185G7 laptop and a Jetson Nano embedded computer. We divided computation time to have higher resolution at shorter computation time regions. It is noted that, as discussed in Subsection 3.2, a lower cost leads to a better trajectory for the robot. The results depicted in Fig. 9 show that the depth-based sampling planner finds a considerably lower cost trajectory than the uniform sampling one, especially in short computation time regions and a less powerful platform.

Specifically, in an embedded computer Jetson Nano, the depth-based sampling method offers a better cost value of -0.7 compared to -0.56 from the uniform technique at the allocated time interval of 20 ms. While with the same computation time interval in an i7 laptop, the best cost value decreases by only 0.03 from -0.96 to -0.99. This fact tells that low-powerful platforms, such as Jetson Nano or other embedded computers for quadrotors, benefit more from the proposed method.

Since we constrained depth sample regions, it is a higher chance for us to sample a collision-free candidate. There-

---

\(^1\) An implementation of the algorithm and the simulation source code are available online at https://github.com/thethaibinh/dess
Therefore, there is a higher ratio of collision-free candidates in the sampled set, and the computation power spent on collision checking decreases, resulting in more sampled trajectories being evaluated as illustrated in Fig. 10. Specifically, depth-based sampling significantly improves the number of evaluated trajectories in the Jetson Nano implementation. At the computation time interval of 20 ms, there are 6969 trajectories evaluated using depth-based sampling compared to only 2546 candidates when the planner samples uniformly. The planner then again has spare time to sample more meaningful trajectories in the space, which include more potential collision-free trajectories as shown in Fig. 11. Remarkably, the proposed roughly doubles the number of collision-free candidates found in the planner running on Jetson Nano, securing an average of 5.12 and 2.25 collision-free candidates in 20 and 6.2 ms, respectively. These results demonstrate that the planner can be implemented and work effectively on Jetson Nano, a practical embedded hardware platform.

In our proposed planner, the computational time for each planning sequence is predefined. The shorter the allocated time for each cycle, the fewer sampled trajectories will be checked for collision. On the other hand, if the time interval is too long, the planning update frequency will be low. Therefore, there is no limit to time allocation as it is the tradeoff between the update frequency and the number of trajectories we will sample in space. Depending on the applications, we must consider the hardware power and the environment’s clutteredness to choose a balanced cycle time. The more cluttered the environment, the more samples must be checked to find a collision-free one.

4.2 Performance Analysis

We evaluate the sampling methods’ efficiency by doing a time complexity analysis. Given a depth image containing $n$ pixels and a computation time, accessing a value from a depth image can be done in constant time. The percentage of obscured-endpoint candidates in the uniformly sampled set and the depth-based sampled set are $RD_{aic}$ and $DB_{aic}$, respectively. And the average numbers of elements $N_{avg}$ in sampled sets in both cases are equal. Checking an obscured endpoint trajectory requires the planner to inflate a new pyramid, which also costs an $O(n)$ operation. The depth-based technique excluded all the obscured-endpoint candidates; thus, $DB_{aic}$ is zero. Since random generators are uniform distributions, $RD_{aic}$ roughly equates the percentage of occupied volume out of the total sample range volume (in the FoV) as illustrated in Fig. 4. In other words, we can call it the level of density $L_{dens}$, thus $RD_{aic} = L_{dens}$. For both uniform and depth-based pipelines, we assume that the ratio $K_c$ of the number of candidates that pass the better-cost checking is equal, e.g., 100 trajectories passing the cost checking to...
Fig. 11 Numbers of generated collision-free trajectories obtained by our proposed depth-based technique and the uniform sampling method used in RAPPIDS [7].

come for collision checking out of 1000 sampled trajectories, following the planning pipeline sketched in Fig. 5. Therefore, the depth-based pipeline will save us an operation of \( O(L_{dens}N_{avg}K_{en}) \) for each sequence. Uniform and depth-based sampling would perform equally in free space because \( L_{dens} = 0 \). But the depth-based method would spare us a substantial amount of computational power in densely occupied structures where \( L_{dens} \) is considerable.

4.3 Setup

We compared planning performance between two policies: the fixed yawing obstacle avoidance method [32] and our proposed planner, which we now call DESS - DEpth-based Sampling and Steering for brevity. The fixed yawing policy utilises a utility function of average velocity to the goal and a collision-checking algorithm with uniform sampling [7], while DESS manipulates a direction function, collision checking with depth-based sampling and an integrated steering mechanism for stuck motions. We evaluated their success rate in navigating the quadrotor to reach a goal.

We conducted experiments for that flight task on a simulation system\(^2\) of a quadrotor flying through an obstacle-dense area from DodgeDrone Challenge 2022 [36]. The challenge is to navigate a quadrotor through the obstacle-dense area to a 65 m far ahead goal. For the convenience of bulk autotests, we decreased the goal distance to 15 m. The obstacle-dense area is a 3D bounding box with dimensions of [0, 15, -5, 5, 0, 10] (meters) in a coordinate system with the axis \( X_O \) and \( Y_O \) parallel to the ground and \( Z_O \) pointing upwards, in which the quadrotor is at [0, 0, 0] and the goal is at [17, 0, 5]. Obstacles are solid spheres of different diameters ranging from 0.1 to 4.0 meters, stationary, and randomly generated inside the bounding box. We generated three obstacle-dense scenarios based on predefined density levels: easy, medium and hard so that there are 29, 51 and 67 obstacles in the bounding box for each scenario, respectively. Figure 12b and c capture a typical scenario with additional visualisation of point cloud ground truth.

All obstacles of the easy scenario are already included in the medium scenario, and the hard scenario has all of the medium obstacles in its configuration. A trial will be labelled successful when the robot reaches the goal’s position before the timeout and does not collide with any obstacle. We integrated the planner with classical nested feedback loops [37] as a low-level controller who received position guidance commands in a receding horizon manner. We limited the maximum desired velocity in the position feedback controller to 1 m/s, mitigating control uncertainty because we only focused on planning evaluation. Whenever a new depth image arrives, if the planner finds a collision-free trajectory, the position controller drops the previous trajectory and starts tracking the new trajectory. Figure 12a describes an overview of the system. We ran the entire system 1000 times for each policy (fixed yawing, DESS) in each scenario (easy, medium, hard) on an i7-1185G7 laptop; thus, we had 6000 flight simulations.

4.4 Results

Whenever the vehicle met a situation of not finding any feasible trajectory (e.g., stuck in front of a huge obstacle or a wall), the DESS planner generated a steering command. Following a steering command, the robot steered consistently until its FoV was clear enough for a normal collision-free trajectory to be generated. Thus, it continued on its mission to the goal. Being equipped with our DESS method, the drone always found a feasible path with success rates for three scenarios are 99.7%, 99.9%, and 99.6%, respectively. In contrast, the simulations using the fixed yawing method often failed when scenarios got more obstacles. Our DESS only failed in a few corner cases due to uncertainties in the position tracking controller. The results summarized in Table 1 prove

\(^2\)Our simulation system source code is available online at https://github.com/thethaibinh/agile_flight

| Table 1 | Success rate of finishing trials by policies |
|---------|---------------------------------------------|
| Fixed yawing [32] | DESS (proposed) |
| Easy | 97.6% | 99.7% |
| Medium | 69.3% | 99.9% |
| Hard | 55.19% | 99.6% |

Bold entries are highlighted results of the proposed method.
the theoretical analysis of DESS’s completeness on convex obstacles.

On the other hand, we also evaluated the finishing time and travel distance for successful trials of the two methods. Figure 13 shows that the average travelled distances and also their standard deviations of our DESS are approximately equal to those of the fixed yawing’s successful trials. This result demonstrates that DESS guarantees robustness while maintaining paths converged. However, it is noted that our method produced much more successful trials than the fixed yawing technique. Figure 14 illustrates that the mean values of DESS’s finishing times are slightly lower than those of the fixed yawing’s successful cases. It can also be noted that standard deviations of DESS are 1.7, 2.0 and 2.7 seconds for the easy, medium and hard, respectively, significantly smaller than those of the fixed yawing, which are 3.1, 4.9 and 5.7 seconds, correspondingly. This comparison shows that there are many trials where the fixed yawing takes a long time to complete, while DESS consistently reaches the goal in more steady amounts of time. That is, DESS is more robust and performs stably in environments of different dense levels. The depth-based sampling gives a head-start in terms of efficiency, allowing DESS to find the best-direction trajectory by evaluating the direction function. Therefore, DESS can find a better direction trajectory, avoiding fluctuated paths caused by lateral motions from the average velocity function.

Figure 15 illustrates a typical executed path for the robot in a hard scenario. It can be seen that the DESS planner is able to effectively generate a collision-free and smooth path for the drone to move to the goal. An illustrated video of how a drone navigates through an obstacle-dense environment to reach a goal can be found in https://github.com/thethaibinh/agile_flight.

On the other hand, we also demonstrate trajectory direction costs along an executed path calculated by the proposed planning algorithm in Fig. 16a. It is noted that when the robot was flying in a free region, it was heading to the goal directly, and the corresponding direction costs were -1. However, when the obstacles were in front of the vehicle, the corresponding costs of the trajectory sequences were slightly higher. In particular, when the vehicle was at a dead end, a steering command was required. In that case, we assigned 1 to the direction cost. Moreover, the proposed depth-based steering did not require additional computing resources. For instance, we summarized the desired yaw (e.g., control input) and corresponding computing time results in a flight trial and showed them in Fig. 16b and c. It can be seen that the computing time for each planning sequence (cycle) remains approximately the same. That is, the proposed direction function paves a concrete platform for the depth-based steering.
algorithm to manipulate, enabling a higher level of autonomy and a better planning performance for our proposed planner compared to the fixed yawing approach.

5 Conclusion

In this paper, we have presented the depth-based sampling method and a local planner that improves memoryless planning efficiency and performance. Our proposed depth-based sampling method exploited available depth data to constrain sampling space, excluding obscured endpoint candidates, which are not meaningful (in-collision) for planning. Thus, the enhanced planner generates a better-cost trajectory over the same amount of allocated computation time compared with the same planner that uses the uniform sampling technique. Equipped with a direction objective function and a depth-based steering technique, the local planner is able to navigate the robot to escape potential stuck motions such as large convex obstacles. It guarantees provably planning completeness, enabling a higher success rate for the planning system in obstacle-dense scenarios.

The simulation results have demonstrated that the proposed approach outperforms the state-of-the-art technique.

This work assumes that the depth estimation and robot’s state estimation accuracy are perfect, and the discussion on it is beyond this project’s scope. These assumptions can be achievable in our proposed simulation environment since both depth input and the vehicle’s state are ground truth.

In future work, we will optimise our solution for implementation in real-world quadrotors, and it requires consideration of depth and robot state estimation uncertainties. While the depth data used in the simulation environment is ground truth, depth input in real applications is subjected to estimation error coming from a commercial depth camera or other sensors with depth estimation engines. In practical applications, depth errors can lead to collisions, decreasing the success rate. We can lower the collision rate by adding safety margins on collision checking. The planner, on the other hand, will then be more conservative, facing more stuck motions. Furthermore, the planner’s performance also relies on the accuracy of global localization. The robot’s localization and orientation variance will affect the trajectory’s direction cost minimization. The degradation of planning...
performance may be increased by frame transformation drift due to the vehicle’s orientation estimation error. These gaps are opportunities for future solutions to be applied.

Currently, the obstacle shape determination is beyond this paper’s scope as we have made assumptions on planning completeness proofs. That means the planner can work normally as it is in convex scenarios without perception of obstacle shape. If it faces a non-convex situation, the planner may get trapped in local minima and cannot guarantee completeness. Obstacle shape perception may be extended in the future considerations.

Since the proposed planner only guarantees instantaneous best direction at every planning sequence, other extensions to this work can be the integration with other techniques to translate it into a local optimal such as the shortest distance traveled or fastest time to finish the course.

Finally, the proposed approach can be expanded to be used with other sensors, such as a monocular camera, where depth data is estimated by learning-based or other methods rather than stereo photography. In that case, it needs to consider the method-based uncertainty in depth estimation.

Appendix

Proof of Lemma 1 All convex obstacles can be represented by the wall and spheroid shapes. When the vehicle hits the boundary of a wall on the motion to the goal, the planner cannot generate any feasible normal trajectory since all sampled candidates lead to collisions. The robot needs to steer so that its camera can have a more open FoV. Assumption 2 manifests that after that steering, the first feasible trajectory’s direction will be parallel to the current side of the wall. And the subsequent consecutive best-cost trajectories would form a line parallel to the wall’s peripheral until they end up at a leaving point $q_L$ (as illustrated in Fig. 7a). Indeed, these trajectories have the best direction costs as they are the minus dot product of two unit vectors of the $m$-line and trajectory’s direction as formulated in Eq. 4. Spheroids and other convex obstacle scenarios can be proved by following the same geometrical approach as visualised in Fig. 7b and c. Thus, when the vehicle faces a convex obstacle, it will reach a leaving point $q_L$ after only the first steer.

Proof of Theorem 1 Assumption 2 shows that the robot will reach $q_S$ by a straight-line path from the start. Once the robot reaches $q_S$, the Lemma 1 ensures that it can also get to a $q_L$ of the current obstacle after only one steer, where it can either go seamlessly to the visible goal or the next obstacle’s $q_S$, as illustrated in Fig. 7d. Assumptions 1 and 2 indicate that once the robot reaches $q_L$ of an obstacle, it never faces that obstacle again in motion toward the goal. Since the environment has finite obstacles, the robot only needs to execute a limited number of steers to reach a $q_L$ where $q_G$ is visible. Assumption 2 again allows the robot to travel directly to $q_G$ from a $q_L$ where $q_G$ is visible. Therefore, the proposed algorithm will always be able to find a feasible path to the goal. □

Acknowledgements This research was supported by the Henry Sutton scholarship from Federation University Australia.

Author Contributions Thai Binh Nguyen conceived original ideas, conducted the research, coding and simulations, and prepared the manuscript. Linh Nguyen primarily supervised the research. He participated in the literature review, discussed the idea, method and results, and edited the manuscripts. Tanveer Choudhury is an associate supervisor. He gave comments on the research and all versions of the manuscript. Manzur Mushid is a co-supervisor. He gave comments on the research and all versions of the manuscript. Kathleen Keogh is an associate supervisor. She gave comments on the research and all versions of the manuscript.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions. Federation University Australia funded the research via the Henry Sutton scholarship - Application ID: 3056759.

Data and/or Code availability All generated data and implementation codes for simulations will be available and maintained in the author’s online repository https://github.com/thethaihabin. If the reader has further needs and questions, do not hesitate to contact the corresponding author.

Declarations

Ethics approval No ethical approval is required by this research.

Consent to participate Not applicable

Consent for publication This paper does not require any consent for publication.

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Popovic, M., Thomas, F., Papatheodorou, S., Funk, N., Vidal-Calleja, T., Leutenegger, S.: Volumetric occupancy mapping with probabilistic depth completion for robotic navigation. IEEE Robot.
2. Dey, R.: Monodepth-vslam: a visual ekf-slam using optical flow and monocular depth estimation. (2021). http://rave.ohiolink.edu/etdc/view?acc_num=ucin1627666226301079. Accessed 1 June 2023

3. Quan, L., Han, L., Zhou, B., Shen, S., Gao, F.: Survey of uav motion planning. IET Cyber-Syst. Robot. 2, 14–21 (2020). https://doi.org/10.1049/iet-csr.2020.0004

4. Florence, P., Carter, J., Tedrake, R.: Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps. Springer Proceed. Adv. Robot. 13, 304–319 (2020). https://doi.org/10.1007/978-3-030-43089-4_20

5. Lopez, B.T., How, J.P.: Aggressive 3-d collision avoidance for high-speed flight in unknown environments. Proceedings - IEEE International Conference on Robotics and Automation, 5759–5765 (2017) https://doi.org/10.1109/ICRA.2019.8793930

6. Li, H., Yang, S.X.: A behavior-based mobile robot with a visual landmark-recognition system. IEEE/ASME Trans Mechatronics 8(3), 390–400 (2003). https://doi.org/10.1109/TMECH.2003.816818

7. Bucki, N., Lee, J., Mueller, M.W.: Rectangular pyramid partitioning using integrated depth sensors (rappids): A fast planner for multicopter navigation. IEEE Robotics and Automation Letters. 5, 4626–4633 (2020). https://doi.org/10.1109/LRA.2020.3003277

8. Florence, P.R., Carter, J., Ware, J., Tedrake, R.: Nanomap: Fast, uncertainty-aware proximity queries with lazy search over local 3d data. Proceedings - IEEE International Conference on Robotics and Automation, 7631–7638 (2018). https://doi.org/10.1109/ARXIV.1802.09076.

9. Herissé, B., Hamel, T., Mahony, R., Russotto, F.-X.: Landing a vtol unmanned aerial vehicle on a moving platform using optical flow. IEEE Trans. Robot. 28, 77–89 (2012). https://doi.org/10.1109/TRO.2011.2163435

10. Maier, J., Humenberger, M.: Movement detection based on dense optical flow for unmanned aerial vehicles. Int. J. Adv. Robot. Syst. 10, 146 (2013). https://doi.org/10.5772/52764

11. Nourani-Vatani, N., Borges, P.V.K., Roberts, J.M., Srinivasan, M.V.: On the use of optical flow for scene change detection and description. J. Intell. Robot. Syst. 74, 817–846 (2014). https://doi.org/10.1007/s10846-013-9840-8

12. Li, H., Yang, S.X.: A behavior-based mobile robot with a visual landmark-recognition system. IEEE/ASME Trans Mechatronics. 8(3), 390–400 (2003). https://doi.org/10.1109/TMECH.2003.816818

13. Silveira, G., Malis, E., Rives, P.: An efficient direct approach to visual slam. IEEE Trans. Robot. 24(5), 969–979 (2008). https://doi.org/10.1109/TRO.2008.2004829

14. Cho, D.-M., Tsiotras, P., Zhang, G., Holzinger, M.: Robust feature detection, acquisition and tracking for relative navigation in space with a known target. American Institute of Aeronautics and Astronautics. ???. (2013). https://doi.org/10.2514/6.2013-5197

15. Loquercio, A., Kaufmann, E., Ranftl, R., Müller, M., Koltun, V., Scaramuzza, D.: Learning high-speed flight in the wild. Sci Robot. 6 (2021). https://doi.org/10.1126/SCIROBOTICS.ABGS810/SUPPL_FILE/SCIROBOTICS.ABGS810_MOVIES_S1_AND_S2.ZIP

16. Kahn, G., Abbeel, P., Levine, S.: Badgr: An autonomous self-supervised learning-based navigation system. IEEE Robot. Automat. Lett. 6(2), 1312–1319 (2021). https://doi.org/10.1109/LRA.2021.3057023

17. Kahn, G., Abbeel, P., Levine, S.: Land: Learning to navigate from disengagements. IEEE Robot. Automat. Lett. 6(2), 1872–1879 (2021). https://doi.org/10.1109/LRA.2021.3060404

18. Nguyen, H., Fyhni, S.H., Petris, P.D., Alexis, K.: Motion primitives-based navigation planning using deep collision prediction, pp. 9660–9667. IEEE. ???. (2022). https://doi.org/10.1109/ICRA46639.2022.9812231. https://ieeexplore.ieee.org/document/9812231/

19. Bircher, A., Kamel, M., Alexis, K., Oleynikova, H., Siegwart, R.: Receding horizon next-best-view planner for 3d exploration. Proceedings - IEEE International Conference on Robotics and Automation. 2016-June, 1462–1468 (2016). https://doi.org/10.1109/ICRA.2016.7487281

20. Oleynikova, H., Burri, M., Taylor, Z., Nieto, J., Siegwart, R., Galceran, E.: Continuous-time trajectory optimization for online uav replanning. IEEE International Conference on Intelligent Robots and Systems. 2016-November, 5332–5339 (2016). https://doi.org/10.1109/IROS.2016.7759784

21. Homung, A., Wurm, K.M., Bennewitz, M., Stachniss, C., Burgard, W.: Octomap: An efficient probabilistic 3d mapping framework based on octrees. Autonomous Robot. 34, 189–206 (2013). https://doi.org/10.1007/S10514-012-9321-0/FIGURES/18

22. Oleynikova, H., Taylor, Z., Fehr, M., Nieto, J., Siegwart, R.: Voxblox: Incremental 3d euclidean signed distance fields for onboard mav planning. IEEE International Conference on Intelligent Robots and Systems. 2017-September, 1366–1373 (2016). https://doi.org/10.1109/IROS.2017.8202315

23. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. (2011) https://doi.org/10.48550/arxiv.1105.1186

24. Cieslewski, T., Kaufmann, E., Scaramuzza, D.: Rapid exploration with multi-rotors: A frontier selection method for high speed flight. IEEE International Conference on Intelligent Robots and Systems. 2017-September, 2135–2142 (2017). https://doi.org/10.1109/IROS.2017.8206030

25. Koltun, V., Scaramuzza, D.: Learning high-speed flight primitives-based navigation planning using deep collision prediction, pp. 9660–9667. IEEE. ???. (2022). https://doi.org/10.1109/ICRA46639.2022.9812231. https://ieeexplore.ieee.org/document/9812231/
for fast and agile exploration using aerial robots. 2020 IEEE International Conference on Robotics and Automation (ICRA), 179–185 (2020). https://doi.org/10.1109/ICRA40945.2020.9196964

34. Kamon, I., Rivlin, E.: Sensory-based motion planning with global proofs. IEEE Trans. Robot. Automat. 13, 814–822 (1997). https://doi.org/10.1109/70.650160

35. Mueller, M.W., Hehn, M., D’Andrea, R.: A computationally efficient motion primitive for quadrocopter trajectory generation. IEEE Trans. Robot. 31, 1294–1310 (2015). https://doi.org/10.1109/TRO.2015.2479878

36. Song, Y., Kaufmann, E., Bauersfeld, L., Loquercio, A., Scaramuzza, D.: Icra 2022 dodgedrone challenge: Vision-based agile drone flight. https://uzh-rpg.github.io/icra2022-dodgedrone/

37. Michael, N., Mellinger, D., Lindsey, Q., Kumar, V.: The grasp multiple micro-uav testbed. IEEE Robotics and Automation Magazine. 17, 56–65 (2010). https://doi.org/10.1109/MRA.2010.937855

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Binh T. Nguyen received the B.Eng. degree in electronics and communications engineering from Hanoi University of Science and Technology, Hanoi, Vietnam, in 2017. He is currently working toward the Doctoral degree with the Centre for Smart Analytics, Federation University Australia. He was with Viettel Aerospace Institute (Hanoi, Vietnam) for four years as an Autopilot Software Developer and collaborated with several other drone/UA V start-ups after that. His research interests include smart navigation for autonomous vehicles, sensor-based motion planning and trajectory generation.

Linh Nguyen (Member, IEEE) received the Ph.D. degree in robotics from the World Leading Robotics Institute, University of Technology Sydney (UTS), Australia, in 2015. Then, he held a Research Fellow position at Nanyang Technological University, Singapore, and UTS, Australia, until early 2020. He is currently a Senior Lecturer at Federation University Australia, Australia. He has authored about 70 highly-ranked international journals and conference papers. His research interests include robotics, artificial intelligence, mobile sensors, and machine perception. Dr. Nguyen was a recipient of the Best Paper Award from the IEEE Innovations in Intelligent Systems and Applications in 2021.

Tanveer A. Choudhury received the Ph.D. degree in engineering from the Swinburne University of Technology, Hawthorn VIC, Australia, in 2014. He is currently a Lecturer of mechatronics engineering at Federation University Australia, Churchill, VIC, Australia. He was with Ericsson for two years as a Radio-Frequency Engineer. His expertise is in applying artificial intelligence and machine learning techniques for predictive modeling and the design and development of control systems. His research interests include a diverse range of disciplines, including materials and surface engineering, mechanical engineering, robotics, and automation.

Kathleen Keogh received the B.Sc. degree (Hons.) from James Cook University, Townsville, QLD, Australia, in 1988 and 1989, respectively, and the Master of Science and Ph.D. degrees from The University of Melbourne, Melbourne, VIC, Australia, in 2002 and 2020, respectively. She is currently a Senior Lecturer in information technology at Federation University Australia, Ballarat, VIC, Australia. Her research interests include organizations, teams, coordination, and using artificial intelligence to support complex decision-making. She is interested in artificial agents working in dynamic situations, the coordination of human and/or artificial team members, and the identification of existing and emergent teams.

Manzur Murshed received a PhD degree in computer science from The Australian National University, Canberra, in 1999. He is currently a Professor of Computer Science in the School of Info. Technology, Faculty of Sci. Eng. & Built Env., Deakin University. Previously, he served Federation University as the Associate Dean Research at the School of Eng., Info. Technology and Physical Sci. (2019–2022), a RHT Smith Professor and Personal Chair in Information Technology (2014–2018), and Monash University as the Head of Gippsland School of Information Technology (2007–2013). He has so far published 240+ refereed research papers with 5,650+ citations (Google Scholar) and h-index 27 & i10-index 83. He has received $5.4m competitive research funding, including four Discovery Projects (DP) grants and a Linkage Infrastructure, Equipment, and Facilities (LIEF) grant from the Australian Research Council (ARC). He has recently served the ARC College of Experts 2020–2022. He has successfully supervised 29 PhD and 3 MPhil students to completion. He served as an Associate Editor for IEEE Trans. Multimedia (TMM) and IEEE Trans. Circuits & Syst. for Video Technol. (TCSVT), two of the top-ranked journals in his fields of research. He received the 2007 Vice-Chancellor’s Knowledge Transfer Award (commendation) from the University of Melbourne, the 2006 Early Career Research Excellence Award from the Faculty of Information Technology, Monash University, and the 1994 University Gold Medal from BUET.