Noname manuscript No.
(will be inserted by the editor)

Challenges and Implemented Technologies Used in Autonomous Drone Racing

Hyungpil Moon[†] · Jose Martinez-Carranza[*] · Titus Cieslewski · Matthias Faessler · Davide Falanga · Alessandro Simovic · Davide Scaramuzza[♣] · Shuo Li · Michael Ozo · Christophe De Wagter · Guido de Croon[♢] · Sunyou Hwang · Sunggoo Jung · Hyunchul Shim[♡] · Haeryang Kim · Min Hyuk Park · Tsz-Chiu Au[♠] · Si Jung Kim

Received: date / Accepted: date

Abstract Autonomous Drone Racing (ADR) is a challenge for autonomous drones to navigate a cluttered indoor environment without relying on any external sensing in which all the sensing and computing must be done on board. Although no team could complete the whole racing track so far, most successful teams implemented waypoint tracking methods and robust visual recognition of the gates of distinct colors because the complete environmental information was given to participants before the events. In this paper, we introduce the purpose of ADR as a benchmark testing ground for autonomous drone technologies and analyze challenges and technologies used in the two previous ADRs held in IROS 2016 and IROS 2017. Six teams which participated in

Hyungpil Moon[†]
Sungkyunkwan University, Seoburo 2066, Jangan-gu, Suwon, Korea
Tel.: +82-31-2994842, Fax: +82-31-2907507, E-mail: hyungpil@skku.edu

Jose Martinez-Carranza[*]
Instituto Nacional de Astrofisica Optica y Electronica (INAOE), Puebla, Mexico, E-mail: carranza@inaoep.mx

Davide Scaramuzza[♣]
University of Zurich, Zurich, Swiss, E-mail: sdavide@ifi.uzh.ch

Guido de Croon[♢]
Micro Air Vehicle laboratory, Faculty of Aerospace Engineering, TU Delft, Delft, the Netherlands, E-mail: g.c.h.e.decroon@tudelft.nl

Hyunchul Shim[♡]
KAIST, Daejeon, Korea, E-mail: hcshim@kaist.ac.kr

Tsz-Chiu Au[♠]
UNIST, Ulsan, Korea, E-mail: chiu@unist.ac.kr

Si Jung Kim
UNLV, Las Vegas, USA, E-mail: si.kim@unlv.edu

these events present their implemented technologies that cover modified ORB-SLAM, robust alignment method for waypoints deployment, sensor fusion for motion estimation, deep learning for gate detection and motion control, and stereo-vision for gate detection.

Keywords Autonomous Drone · Drone Racing · Autonomous Flight · Autonomous Navigation

1 Introduction

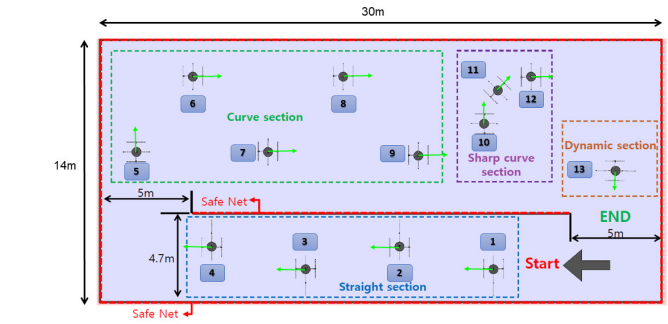
The Autonomous Drone Racing (ADR) was inaugurated in IROS 2016 Daejeon, Korea and continued in IROS 2017 Vancouver, Canada. These ADRs were to help advancing the pilot-less autonomous navigation of an unmanned aerial vehicle in indoor racing tracks which contained five testing elements: a high-speed flight on a straight path through open gates, sharp turns, horizontal zig-zag path, a spiral upward path through closed gates, and a dynamic obstacle. As a brief summary of the two competitions, there were 4 open gates, 22 closed gates, and total 26 including one dynamic gate in ADR 2016. To facilitate the localization, each track gate had a QR code that contained the identification number of the gate in ADR 2016. The racing track in ADR 2017 was revised: the open gates were replaced by tree-like obstacles and the 360 degree-spiral-up gates in ADR 2016 was replaced by 90-degree-spiral-up gates. The new tree-like open gates were for testing faster flight in a straight path and the number of closed gates was reduced to 9 including one dynamic gate. The detailed information of the ADR track is available online [1].

The map information was given to participants prior to the competitions and the racing track was prepared as close to the CAD model as possible in the two competitions. Therefore, the main purpose of these settings was to test the autonomous navigation capability of drones based on onboard sensors and computation only. Technically, the challenges for drones were stable flight control, robust detection of gates, registration of gates in the path planning, tracking control, and detection of the dynamic obstacle. Although the individual problem may be considered to be easily solvable for computationally and sensorily rich systems, ADR is not the case due to the limited resources.

In 2016, 11 teams registered the competition, but only three teams (Team KIRD of KAIST Korea, Team MAV-lab of TU Delft, Netherlands, and Team ETHZ ASL/ADRL, Switzerland) finally competed in the arena. The summary of ADR 2016 can be found in [1].

In 2017, 14 teams registered the competition and 7 teams (Team KAIST Korea, Team MAV-lab of TU Delft, Netherlands, Team QuetzalC++ of INAOE, Mexico, Team First Commit of enthusiasts in Bay-area, Team Drone bot of UNIST, Korea, Team Robotics and Perception Group of Univ. of Zurich, Switzerland, Team LOBO DRONE of Univ. New Mexico, USA) came to the

[1] ADR2016: http://rise.skku.edu/home/iros2016racing.html, ADR2017: http://ris.skku.edu/iros2017racing/



(a)

(b)

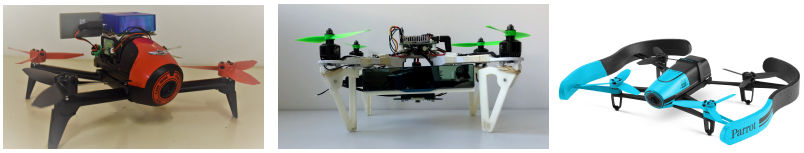
Fig. 1 (a) Top view of the schematic representation of the arena for the Autonomous Drone Racing, the green arrows indicate the direction in which the drones had to fly through the gates; (b) Examples of the gates placed across the arena.

competition but only 5 teams finally competed in the arena. In 2017, teams were more experienced and organized and implemented more advanced technologies for the competition. The team first commit reached gate 8 (which is the fourth closed gate) at 01:56.5. Team MAV-lab reached gate 7 at 00:25.7. Team Robotics and Perception Group reached gate 8 at 00:35.8. Team QuetzalC++ reached gate 9 at 03:11.6 and won the competition.

In the following sections, the methodologies of participating teams are presented. Gate detection methods of teams that could not compete in the racing track have been included since these teams presented their work when registering. Since the flight control for conventional quadrotors is considered as a solved problem, the dynamic modeling and control issue is not discussed in this paper. In Sec. 2, drone hardware systems of successful teams are described. In Sec. 3, the winner of ADR 2017, team INAOE's strategy is presented. They implemented waypoint tracking with ORB-SLAM without an explicit scaling measure. In Sec. 4, team Robotics and Perception Group of UZH presents their waypoint tracking method using onboard depth sensor information. In Sec. 5, team MAV-LAB of TU Delft presents high-level navigation method using a state machine and low-level sensor fusion method for position and velocity

estimation. Team KAIST and team UNIST present a deep learning application for the gate detection in Sec. 6 and for end-to-end learning in Sec. 7, respectively.

2 Aerial Vehicles



(a) (b) (c)

Fig. 2 Drone hardwares. (a) INAOE's Parrot™ Bebop 2 (b) UZH RPG' drone. (c) TU Delft's Parrot™ Bebop 1.

Team INAOE used the Parrot® Bebop 2.0 Drone (see Fig. 2(a)). This vehicle can transmit inertial and visual data via WiFi. Visual data is captured from an on-board camera with an image resolution of 840×480 pixels transmitted at 30 Hz; inertial data is captured with an on-board Inertial Measurement Unit (IMU) transmitting at 5 Hz; altitude measurements are transmitted at 20 Hz. Communication and programming of control commands with the Bebop 2.0 are possible thanks to the Software Development Kit (SDK) known as *bebop autonomy*, released by the Parrot® company and made available as a ROS package. However, programs can not be run on board the vehicle. It should be noticed that the competition rules requested processing on-board exclusively, therefore, participating with the Bebop 2.0 was not an option. To enable on-board processing, team INAOE adapted an Odroid computer model XU4, equipped with an octa-core processor running at 2.0 GHz, with 2 GB in RAM, and with a WiFi module. To compensate for the weight added by mounting the Odroid on the vehicle, the battery case was removed. The Odroid was also powered by the vehicle's battery. Fig. 2(a) shows the Bebop 2.0 with team INAOE's adaptations, which were accepted as valid to participate in the competition. Linux version Ubuntu 16 LTS ran as the operating system on the Odroid. The Robotic Operating System (ROS) also ran on the Odroid for software communication and implementation, including INAOE's metric monocular SLAM solution.

Team UZH RPG's drone is shown in Fig. 2(b) which is a DJI F330 frame, while CM2208/2000 Cobra motors with stiff six inches propellers provided actuation. It is equipped with (1) the Qualcomm Snapdragon Flight board equipped with a large field-of-view camera used for Visual-Inertial Odometry, used for state estimation, (2) the Intel RealSense RGB-D camera, connected to the Up-Board computer through USB., (3) the Up-Board computer, used

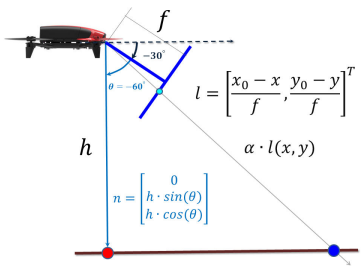


Fig. 3 Geometric configuration used by team INAOE to generate a synthetic depth image, which is coupled with an RGB image, captured with the onboard camera, to be used by ORB-SLAM in its RGB-D version, thus obtaining pose estimates with metric.

for planning, control and map alignment, (4) the Lumenier F4 AIO flight controller. The two onboard computers communicated through UART to provide UZH's high-level position control system with the state estimate of the vehicle. The output of such high-level control, namely the reference collective thrust and body rates, were sent to a Lumenier F4 AIO flight controller, which was responsible for motor control. The quadrotor had a take-off weight of 950 g and motor-to-motor diagonal of 330 mm.

Team TU Delft used a Parrot® Bebop 1 (shown in Fig. 2(c)). It is a $33 \times 38 \times 3.6$ cm drone (with the outer hull) that is equipped with a downward facing narrow-view camera and a forward facing fish-eye camera, an Inertial Measurement Unit (gyros, accelerometers, magnetometers), a pressure-meter, and a downward facing sonar. For the race, team TU Delft fully replaces the standard software on-board of the Bebop with the Paparazzi autopilot code [2]. The Paparazzi software[2] controls all aspects of the drone, from reading and processing all sensor data and images to controlling the rotors. All the processing for the drone race took place on the Parrot P7 dual-core CPU Cortex 9 (max 2GHz).

3 INAOE's approach: Monocular Metric Visual SLAM for Autonomous Flight

INAOE's strategy to address the competition challenge was based on two main components: 1) PID controllers to control height, heading and forward/sideways motions; 2) drone's localization based on a metric monocular SLAM. Team INAOE did not use or build a 3D map of the arena before its participation in the race. Instead, INAOE used a relative waypoint system where the controller would navigate the drone towards a waypoint whose position was relative to the previous one.

[2] `http://wiki.paparazziuav.org/`

3.1 Metric Mono SLAM Assuming a Planar Ground

Autonomous navigation of a drone in the indoor competition arena posed a challenging scenario. External methods for localization of the drone, such as motion capture systems or fiducial markers, were not allowed. To overcome this restriction, team INAOE decided to use a visual SLAM method for localization of the drone.

Given that the Bebop 2.0 has an on-board monocular camera, ORB-SLAM [3] was employed to obtain the camera pose and 3D point estimates with the caveat that ORB-SLAM generates without metric when used with a monocular camera, however, INAOE's solution based on the waypoint system described before relies on the relative position of the waypoints given in meters.

To address the scale problem, team INAOE exploited the fact that the ground in the arena was planar. Thus, by assuming a planar ground, and by knowing the camera angle and drone's height, a synthetic depth image was generated by resolving a ray-plane intersection geometry. The synthetic depth image was coupled with incoming RGB frames and used in the RGB-D version of ORB-SLAM, which generates metric pose estimates. Two functionalities offered by the Bebop 2.0 were exploited for the race competition: 1) the onboard camera in the Bebop 2.0 is a fish-eye camera whose field of view is foveated, via software, to produce a rectangular image and this foveation means that the camera may point forward at an angle of zero degrees or look downwards at an angle of up to -85^o, and such angle can be controlled via the Bebop's SDK, the resulting image is also gyro-stabilized; 2) the Bebop 2.0 provides altitude via barometric and ultrasound for lower altitudes, these altitude measurements tend to be accurate within centimeters, depending on the ground material.

The above features were combined with metric for the pose estimates. For the latter, team INAOE extended their previous work [4] to generate synthetic depth images based on the line-plane intersection problem by formulating a geometric configuration where it is assumed that *the ground is planar*. In addition, the Bebop's altimeter is used to obtain an estimate of the camera's height h, the camera angle read through the SDK is used to calculate the angle at which a vector n would be located with respect to the origin in the camera's coordinate system with length h. This vector n is perpendicular to the planar ground, hence it can be used to know a point lying on this planar ground with normal n. Therefore, for each pixel at coordinates (x, y) on the image, a vector l departing from the camera's optical center (x_0, y_0) and passing through the pixel at (x, y) will intersect the planar ground for some scalar α. Fig. 3 illustrates a side view of this geometric configuration for the case when the bebop's camera is foveated to the angle of -30^o with respect to the horizon. The line-plane intersection equations are used to find α, thus

obtaining the 3D point at which l intersects the ground plane by calculating:

$$l = \left[\frac{x_0 - x}{f}, \frac{y_0 - y}{f}, 1\right]^\top \tag{1}$$

$$\theta = -(90 - 30) \tag{2}$$

$$n = [0, h \cdot \sin(-\theta), h \cdot \cos(-\theta)]^\top \tag{3}$$

$$\alpha = \frac{n^\top \cdot n}{l^\top \cdot n} \tag{4}$$

$$\tag{5}$$

Hence, α is the depth corresponding to the pixel (x, y).

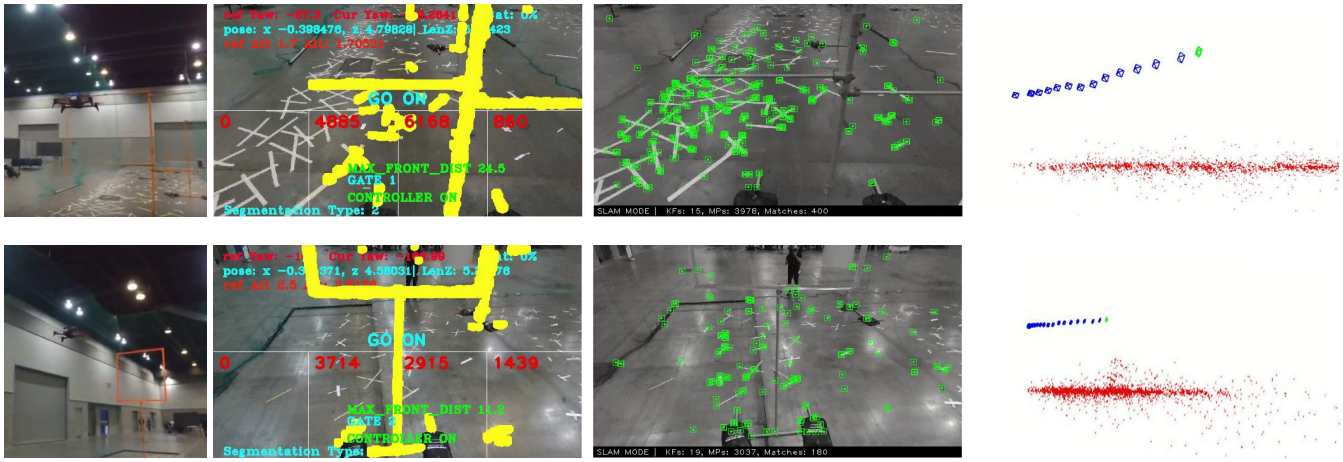


Fig. 4 Snapshots that illustrate the performance of team INAOE using their metric monocular SLAM system during the race; gate detection is also carried out using color segmentation.

3.2 Autonomous Navigation Approach

The dimensions and approximate positions of the gates were known in advance, therefore, team INAOE defined a set of waypoints with the format $\mathbf{w}_i = [h_{ij}, \psi_{ij}, L_{ij}, F_{ij}]$, which indicates that once the drone has successfully reached the waypoint $\mathbf{w}_i$ then it has to change its height and heading, and it has to fly forward or sideways towards the next waypoint $\mathbf{w}_j$, the decision on having to fly forward or sideways was indicated by the field F_{ij}. In this sense, h_{ij} is the reference height in meters, ψ_{ij} is the reference heading in degrees, L_{ij} is the reference length in meters, and F_{ij} may take one of three values: 0 - fly forward; 1 - fly sideways to the left; 2 - fly sideways to the right. Waypoints were distributed along the arena forming a trajectory such that the drone had

to follow a straight line from gate 1 to 4, then fly through gate 5 by changing orientation and height, and from 6 to 9, move sideways and then forward to fly through the corresponding gates. Similar waypoints were planned for gate 10 to 13.

The control was implemented with 3 controllers based on the PID (Proportional-Integral-Differential) controller to control height, heading (yaw controller) and forward/sideways motion (pitch and roll controller), trigger in that order once the drone reached a waypoint, current height and yaw was measured with the drone's altitude and IMU sensors. The forward/sideways motion controller would fly the vehicle forward/sideways aiming at reaching the desired length, this length was measured as the magnitude of the drone's position obtained with INAOE's metric monocular SLAM. In addition, during the forward/sideways motion and from gate 5 onwards, the gate detection based on color segmentation was used to detect the gate's pole base, whose position relative to the image center generates an error that was used to correct drift in the direction of the drone with respect to the current gate.

Team INAOE evaluated their metric monocular SLAM approach using the Vicon motion capture system in indoor environments with a planar ground. It was found that the pose error is of **2**% on average. Fig. 4 shows examples of INAOE's whole system carrying out gate detection and metric mapping.

3.3 Discussion and Limitations

INAOE's drone managed to fly through **9 gates** in its first time slot and 5 gates in its second time slot. The 9^{th} gate was reached at the time of 3 minutes with 11.6 seconds, however, INAOE's drone speed was estimated to be of 0.7 m/s. The reason why the drone took more time in its flight was due to the yaw controller, which spent considerable time in reaching the yaw reference, for instance, right at the outset, the controller spent 25 seconds to reach the first yaw reference. This was due to a non-optimal tuning of the yaw controller and possibly also due to the low frequency of the IMU (5 Hz).

Furthermore, the image segmentation was also used to masked the gates and removed them from the synthetic depth image in order to avoid incorrect initialization of feature and map corruption. The gate detection was also used to correct the drone's heading in those cases where it began drifting with respect to the gate. However, the contribution of this error in the controllers was tuned with a low gain to avoid unstable controlling. This controller will be improved in future work.

4 UZH's approach: CAD model-based Localization

The strategy for team UZH was based on the availability of a CAD model of the race track to assign suitable waypoints along the track and used onboard visual odometry to drive the quadrotor through the race. The main challenge

in UZH's approach was to robustly align the track reference frame, where the waypoints were placed, with the odometry reference frame. To solve this problem, an onboard depth sensor was used altogether with map alignment by minimizing the distance between the expected pointcloud of the race track and the one provided by the sensor.

4.1 Nomenclature

Let W be the world reference frame and G the reference frame of a gate in the track. Without the loss of generality, it is assumed the origin of G to be placed at the center of the aperture of each gate. The position and the orientation of G with respect to W are defined by ${}_W\mathbf{p}_G$ and $\mathbf{R}_{WG}$, respectively. Let O be the odometry reference frame. The position and the orientation of the quadrotor's body frame B with respect to W are defined by ${}_W\mathbf{p}_B$ and $\mathbf{R}_{WB}$, respectively. The relative transformation $\mathbf{T}_{WO}$ between W and O is represented by the translation vector $\mathbf{t}_{WO}$ and the rotation matrix $\mathbf{R}_{WO}$. Finally, let G be the reference frame of a gate in the track.

4.2 Strategy

As previously mentioned, the strategy of team UZH relied on the availability of a CAD model of the race track. More specifically, such model provided the position ${}_W\mathbf{p}_G$ and the orientation $\mathbf{R}_{WG}$ of each gate in the frame W. Therefore, it was possible to define waypoints in that frame such that the quadrotor could fly through them to perform the race. For the straight section, four waypoints were selected: two in the center of the aperture of gates 1 and 3, respectively; two on the left of gate 2 and 4, along the straight line connecting the previous two waypoints. For the rest of the track, a waypoint was placed in the center of the aperture of each gate to be traversed (cf. Fig. 1). Additionally, for each waypoint, the velocity vector was defined such that the gate was traversed orthogonally. While flying from one waypoint to the next, the heading of the quadrotor was controlled such that an on-board depth camera always faced the next gate to be traversed.

Using the approach described above, it was possible to fully define a sequence of waypoints the robot was supposed to navigate through in order to accomplish the race. However, such waypoints were defined in the reference frame W, while the robot, thanks to an on-board Visual-Inertial Odometry pipeline, was aware of its position, orientation and velocity with respect to the odometry frame O. An initial guess of the transformation $\mathbf{T}_{WO} = (\mathbf{t}_{WO}, \mathbf{R}_{WO})$ was obtained during the test days by fixing the starting position of the drone, i.e. the origin of O, and manually measuring $\mathbf{T}_{WO}$. Nevertheless, small errors in the relative orientation between the two frames, as well as drift in the Visual-Inertial Odometry, could potentially lead to crashes with the gates or the protection nets around the track. This made it necessary to improve the

initial guess of $\mathbf{T}_{WO}$ by estimating it online in order to correctly align the odometry frame with the world frame.

4.2.1 Frames Alignment

To perform the frame alignment, team UZH used an Iterative Closest Point (ICP) algorithm [5]. UZH's quadrotor was equipped with a front-looking depth camera providing a pointcloud $\mathcal{P}_{C^\star} = \left\{\mathbf{p}_{C^\star i} \in \mathbb{R}^3\right\}$, expressed in the true camera frame $C^\star$, of the surroundings of the vehicle. Also, a pointcloud $\mathcal{Q}_W = \left\{\mathbf{q}_{Wi} \in \mathbb{R}^3\right\}$, expressed in the world frame, of the race track was obtained from the CAD model. Based on these two point clouds, the current VO pose estimate $\mathbf{T}_{OB}$ and the current alignment estimate $\mathbf{T}_{WO}$, ICP was used to estimate the true pose $\mathbf{T}^\star_{WB}$. First, the point cloud observation that is expected for $\mathcal{P}_{C^\star}$ given the current estimate was obtained as follows:

$$\tilde{\mathcal{Q}}_C = ((\mathbf{T}_{WO} \cdot \mathbf{T}_{OB} \cdot \mathbf{T}_{BC})^{-1} \cdot \mathcal{Q}_W) \cap \mathcal{F}_C \tag{6}$$

where $\mathcal{F}_C \subset \mathcal{R}^3$ is the *view frustum* [6] of the depth camera, representing the camera-centered volume in which it can accurately detect depths. Given two point clouds $\mathcal{P}_A$ and $\mathcal{P}_B$, ICP returns a relative transform $\mathbf{T}_{AB}$ that best satisfies $\mathcal{P}_A \sim \mathbf{T}_{AB} \cdot \mathcal{P}_B$. Thus, by passing $\mathcal{P}_{C^\star}$ and $\tilde{\mathcal{Q}}_C$ to ICP, it is obtained an estimate for the camera frame correction $\mathbf{T}_{C^\star C}$ that aligns the measured and expected point clouds. Given this, in theory $\mathbf{T}_{WC} = \mathbf{T}_{WC^\star}$ could be directly solved for an updated $\mathbf{T}_{WO}$, but this would be prone to noise in ICP. Thus, estimates for $\mathbf{T}_{C^\star C}$ were accumulated over the k most recent ICP measurements. Therefore, let $\{\mathbf{T}^1_{OB}, \mathbf{T}^2_{OB}, ..., \mathbf{T}^k_{OB}\}$ be the corresponding odometry pose estimates and $\{\mathbf{T}^1_{C^\star C}, \mathbf{T}^2_{C^\star C}, ..., \mathbf{T}^k_{C^\star C}\}$ the corresponding camera pose correction estimates provided by ICP. Then, $\{\mathbf{T}^i_{WC} = \mathbf{T}_{WO} \cdot \mathbf{T}^i_{OB} \cdot \mathbf{T}_{BC}\}$ and $\{\mathbf{T}^i_{WC^\star} = \mathbf{T}^i_{WC} \cdot (\mathbf{T}^i_{C^\star C})^{-1}\}$ are the corresponding estimated and "true" camera poses. Given this, $\mathbf{T}_{WO}$ is updated using nonlinear least squares such that it minimizes the distance between estimated and "true" camera positions:

$$\mathbf{T}_{WO} \leftarrow \text{argmin} \sum_{i=1}^{k} \omega_i \left\| \mathbf{p}^i_{WC} - \mathbf{p}^i_{WC^\star} \right\| \tag{7}$$

Where ω_i is a time decay which assigns higher weight to newer measurements. Two precautions were taken to avoid degenerate $\mathbf{T}_{WO}$: firstly, (7) was constrained in roll and pitch by restricting $\mathbf{T}_{WO}$ to rotate only in yaw. This can be done given that the VIO system can observe gravity and thus has no drift in roll and pitch. secondly, $\mathbf{T}_{WO}$ is only updated according to the above formula if there is sufficient baseline between $\mathbf{p}^1_{OB}$ and $\mathbf{p}^k_{OB}$. Otherwise, (7) is not constrained in yaw.

4.2.2 Planning

Let ${}_W\mathbf{p}_G$ and $\mathbf{R}_{WG}$ be the position and the orientation with respect to the frame W of a gate the quadrotor has to traverse. Using the estimate of $\mathbf{T}_{WO}$

obtained as described in Sec. 4.2.1, it is possible to transform such quantities into the odometry frame as:

$$ {}_O\mathbf{p}_G = \mathbf{R}_{WO}^{\intercal} \left({}_W\mathbf{p}_G - {}_W\,\mathbf{t}_{WO} \right) \tag{8} $$

$$ \mathbf{R}_{OG} = \mathbf{R}_{WO}^{\intercal} \mathbf{R}_{WG}. \tag{9} $$

Let $\hat{n}$ be the unit vector orthogonal to the gate (i.e., the first column of $\mathbf{R}_{OG}$) expressed in the odometry frame, and let ${}_O\mathbf{p}_B$ the position of the quadrotor in the same frame. The position error is defined at time t_k as $\mathbf{e}\left(t_k\right) =_O \mathbf{p}_B\left(t_k\right) -_O \mathbf{p}_G$, and decompose it into the longitudinal error $\mathbf{e}_{\text{lon}}\left(t_k\right)$ and the lateral error $\mathbf{e}_{\text{lat}}\left(t_k\right)$ as:

$$ \mathbf{e}_{\text{lon}}\left(t_k\right) = \mathbf{e}\left(t_k\right) \cdot \hat{n} \tag{10} $$

$$ \mathbf{e}_{\text{lat}}\left(t_k\right) = \mathbf{e}\left(t_k\right) - \mathbf{e}_{\text{lon}}\left(t_k\right). \tag{11} $$

The velocity feedback input $\mathbf{v}$ is defined as:

$$ \mathbf{v}\left(t_k\right) = K_{\text{lon}}\mathbf{e}_{\text{lon}}\left(t_k\right) + K_{\text{lat}}\mathbf{e}_{\text{lat}}\left(t_k\right), \tag{12} $$

where K_{lon} and K_{lat} are diagonal, positive definite gain matrices.

Let $\mathbf{p}_B^d\left(t_{k-1}\right), \dot{\mathbf{p}}_B^d\left(t_{k-1}\right), \ddot{\mathbf{p}}_B^d\left(t_{k-1}\right)$ be the desired position, velocity and acceleration (i.e., the reference quadrotor state) at time t_{k-1}, expressed in the odometry frame. Let $\tilde{\mathbf{v}}\left(t_k\right)$ be a low-pass filtered version of $\mathbf{v}\left(t_k\right)$, *i.e.*, $\tilde{\mathbf{v}}\left(t_k\right) = \alpha\tilde{\mathbf{v}}\left(t_{k-1}\right) + \left(1-\alpha\right)\mathbf{v}\left(t_k\right)$. The reference state at time t_k is computed as:

$$ \ddot{\mathbf{p}}_B^d\left(t_k\right) = \frac{\tilde{\mathbf{v}}\left(t_k\right) - \dot{\mathbf{p}}_B^d\left(t_k\right)}{\Delta t} \tag{13} $$

$$ \dot{\mathbf{p}}_B^d\left(t_k\right) = \tilde{\mathbf{v}}\left(t_k\right) \tag{14} $$

$$ \mathbf{p}_B^d\left(t_k\right) = \mathbf{p}_B^d\left(t_{k-1}\right) + \tilde{\mathbf{v}}\left(t_k\right)\Delta t, \tag{15} $$

where $\Delta t = t_k - t_{k-1}$ and $\alpha \in [0,1]$.

Once the reference position $\mathbf{p}_B^d\left(t_k\right)$ is known, the reference yaw angle $\Psi^d\left(t_k\right)$ (i.e., the heading of the vehicle) can be computed, pointing towards the gate to be traversed, and the yaw rotational speed $\dot{\Psi}^d\left(t_k\right)$. Thanks to the fact that the depth camera is front-looking, the direction it has to point toward is defined by the vector $\mathbf{u} = [u_x, u_y, u_z]^{\intercal} =_O \mathbf{p}_G - \mathbf{p}_B^d\left(t_k\right)$, and the reference yaw angle is that $\Psi^d\left(t_k\right) = \text{atan2}\left(u_y/u_x\right)$.

Team ETH used the position error $\mathbf{e}$ to check whether the robot reached the desired waypoint and, if this is the case, then the drone was moved to the one for the next gate according to the sequence determined by the rules of the competition. If all the gates in the track have been traversed, the quadrotor would have been commanded to safely land.

4.2.3 Control

To track the reference state $\mathbf{x}^d\left(t_{k-1}\right) = \left[\mathbf{p}_B^d\left(t_{k-1}\right), \dot{\mathbf{p}}_B^d\left(t_{k-1}\right), \ddot{\mathbf{p}}_B^d\left(t_{k-1}\right), \Psi^d\left(t_{k-1}\right)\right]^\intercal$, team ETH used the control strategy reported in [7]. Broadly speaking, the control pipeline can be split into a *high-level* and a *low-level* component. The high-level controller receives as input the reference position, velocity, acceleration and yaw, and produces the desired collective thrust and body rates. These are sent to the low-level controller, which is responsible for body rate control (i.e., transforms the reference body rates into desired torques) and computes the single-rotor thrusts necessary to achieve the reference collective thrust and torques.

4.2.4 Dynamic Obstacle

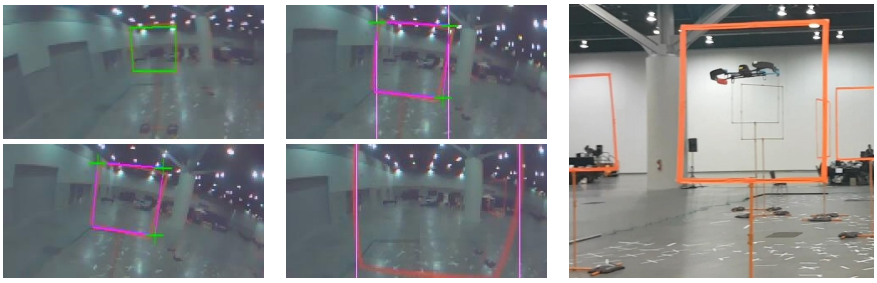
Team ETH's strategy for the dynamic obstacle envisaged that the quadrotor would stop in front of it at a predefined distance, in order to detect the moving bar by exploiting the pointcloud provided by the front-looking camera. Once enough detections of the bar were obtained, the quadrotor would plan a trajectory passing through the center of the bottom half of the gate, such that the quadrotor would be traversing the gate when the bar was pointing straight up. The duration of the trajectory was computed given the knowledge of the rotational velocity of the bar and the distance to it at the start of the maneuver.

5 TU Delft's approach: State Machine based High Level Navigation

Team TU Delft's main goal in the autonomous drone races at IROS is to create and demonstrate small drones that are able to move at high speed in their environment. According to team TU Delft, 'small' to them means < 50 cm for now, but an approach that can downscale even to < 15 cm drones will be intended. Aiming for such small sizes entails important limitations in on-board sensing and processing, and, consequently, the artificial intelligence and control algorithms used onboard [8]. Specifically, it means to focus on monocular navigation and computationally extremely efficient algorithms for robotic vision and control.

5.1 High-level navigation

The drone has to successfully pass through the gates in the exact order as determined by the drone race organization. Hence, the drone needs to navigate autonomously to specific places in the environment. For navigation, team TU Delft chose not to employ highly accurate but computationally complex SLAM or Visual Odometry (VO) methods. Instead, they opted for a state machine where each state represented the behavior during a part of the track. For



(a) (b) (c)

Fig. 5 TU Delft's gate detection consisted of two strategies running at the same time. (a) When the entire gate is visible in the image, a snake-gate detector was designed and used.(b) In parallel, a vertical histogram of colors based approach would select the most likely vertical edges of the closest gate. This allows tracking to continue even when part of the gate was occluded. (c) TU Delft drone passing a gate autonomously.

example, the first state had the drone take off, pitch forward, and then fly straight for the first 12m of the track. After covering 12m, the next state had the drone turn 90 degrees to the right and subsequently pass through the closest gate in view. All states consisted of linked sub-behaviors, such as climbs, descents, coordinated turns, sideways motion, and gate-pass-throughs. These sub-behaviors required rough odometry estimates and the determination of the position and orientation of the gates - as will be detailed in the next subsection.

The advantage of the developed state-machine-based high-level navigation is that it is computationally extremely efficient and can easily accept changes in gate positions. The disadvantage is that the drone will not be able to recover if its position is too far off from the expected nominal position.

5.2 Low-level sensing and control

The sub-behaviors mentioned above required two main pieces of information: the position and orientation of the drone with respect to the gate and the velocity of the drone.

The gate is detected by means of a 'snake-gate' algorithm that only processes small parts of the image. The detection is fully color-based, which means a reliance on the orange gates being visible in the small and blurry 315×160 pixel images (See Fig. 5). The detected corners of a gate, in combination with its known geometry, allow the drone to determine its position and orientation with respect to the gate. This provides the drone with the necessary position offsets to pass through gates, but also gives very accurate velocity estimates when a gate is in sight. The fact that gate detections are actively used to guide the drone also means that moving the gates slightly forms no problem for the navigation, as long as the gate is visible from the position where the drone expected to be able to see a gate, typically 3 m in front of the gate.

There are large parts of the track where the drone would not see a gate, while it still needs velocity estimates for odometry and control. At IROS 2017, team TU Delft aimed to reach a considerable speed with the drone. Initial tests at higher speeds showed that the standard solution of combining sonar with optical flow from the bottom camera significantly degraded due to the blur in the images, even with abundant texture on the floor. Hence, team TU Delft opted for not using the bottom camera, and instead rely on knowledge of the drag of the airframe to estimate the velocity. The drag is estimated by means of a drag model that uses the accelerometer measurements as inputs.

Both the position and velocity estimations from the gate detection and the velocity estimates from the drag model are combined in an Extended Kalman Filter (EKF), which also estimated the biases of the accelerometers.

5.3 Results

The gates as initially foreseen in the competition were hardly visible in the 315×160 pixel onboard images of the drone in the low light condition of the basement hall where the competition was held. Prior to the competition, the organizers adjusted the gates to be more visible by adding bright orange tape to the gates. From that point on, the team focused on the fine-tuning of the control strategy. While in the initial practice runs, team TU Delft passed through 7 and then even 8 gates quite easily, for some reason, the drone would start steering erratically after passing gate 6 and sometimes even gate 5. It was not possible to find the cause of this phenomenon before the competition.

During the competition, TU Delft's drone flew through 7 gates in 25 seconds, which made it the fastest drone on the track. However, other teams managed to gate 8 and 9, which resulted in the fourth place of the team. Later in the track, the drone made mistakes such as detecting a gate successfully but then steering in completely the wrong direction. Post-competition analysis showed that there was a bug in the bias estimation of the Kalman filter, leading to diverging bias and velocity estimation after the drone had turned 180 degrees. Any time the drone would fly in the initial direction again, the filter would quickly converge again, which explains why the problem was not observed during the testing in a smaller test area before the competition. These unstable biases caused the incorrect decisions on the part of the drone, steering the wrong direction while the gate detections - as apparent from the onboard imagery in Fig. 5 - were correctly detected.

6 KAIST approach: Detection based Strategy

The key idea of team KAIST's approach was to detect the nearest gate using a single onboard camera and fly through it while using the information of the general layout of the arena. The rest of this section describes the key points of team KAIST's strategy and discusses their strengths and weaknesses.

6.1 Strategy

Team KAIST's approach is pivoted on the reliable gate detection. When the center of the nearest gate is detected, a waypoint-based guidance algorithm allows for the drone to fly through the center. After passing the gate, based on the given map, it looks for the next gate using the given map. Therefore, an accurate gate detection is the key point in the strategy of team KAIST. In IROS 2016, Team KAIST used a color-based detection method[9], which was found too sensitive to illumination changes. Therefore, for more reliable detection, Jung et al. introduced ADRNet[10], which is a deep learning based detection method. The following subsection briefly presents ADRNet and LOS guidance.

6.1.1 Gate detection using ADRNet

For ADR, it is required to detect the gate using onboard sensors in real time. A novel deep convolutional neural network detection model, named ADRNet, was proposed for the real-time image processing. ADRNet is based on the SSD (Single Shot multibox Detector)[11] architecture. To improve speed, the base network structure was changed from VGG-16[12] to AlexNet[13]-like network with seven convolutional layers and removed two high-level feature layers. Anchor box sizes and feature extraction points were also modified to improve accuracy. ADRNet achieved inference speed of 28.95 fps(frames per second) on a NVIDIA TX2 embedded board. The detection rate of ADRNet on a gate detection dataset was 85.2% and average precision was 0.755.

6.1.2 LOS Guidance

A line of sight (LOS) guidance algorithm was adopted for precise maneuver through the gate center. This algorithm is frequently used for fixed-wing aircraft landing. This algorithm is slightly modified for quadrotor dynamics, which has decoupled dynamics between roll and yaw axis [10].

6.2 Pros and Cons

The team KAIST's detection-based approach has a clear advantage that it can be applied to situations with higher uncertainties because it does not heavily depend on the prescribed map. The following subsection presents the strengths and weaknesses of the detection based approach.

6.2.1 Pros

ADRNet shows a robust detection performance, which is far better than the previous approach used in IROS 2016. A gate of ADR 2017 arena is shown

(a) (b)

Fig. 6 Gate detection by KAIST (a) A racing gate of the ADR 2017 arena (left) and gate detection result using ADRNet (right) (b) Gate detection results on various backgrounds

in Fig. 6(a). The ADR arena was quite dim and cluttered with various background objects. Nonetheless, ADRNet showed good gate detection results. As data accumulates, the deep learning based detector becomes more robust against background and lighting conditions. With a sufficiently large dataset has been constructed in various occasions, ADRNet can be applied to many places without further training. The detection results in various conditions are shown in Fig. 6(b).

6.2.2 Cons

The detection based approach is inefficient if the environment is fully mapped. It is especially true for ADR, where the drone needs to fly as fast as possible for a higher score. Also, as the Deep neural networks require a higher computing power, which implies a heavier computer with a larger battery, which makes the drone slower and less maneuverable. Another minor problem is that ADRNet needs to be retrained if the environment is significantly different from the existing dataset. This poses a logistics problem, which can quite negatively impact the team's performance.

In summary, the detection based approach can perform drone racing without a precise map or in a dynamic environment with moving gates. However, there are shortages such as inefficiency of the two-step approach and too much labor for constructing dataset.

7 UNIST's Approach: End-to-end Deep Learning

Upon receiving an image from an input video stream, the image is fed into the deep neural networks, which will then generate control commands for different aspects of the control, including horizontal actions, vertical actions, and rotational actions. Notice that all three neural networks take the same input video stream simultaneously. The output of the neural networks is combined to form a control command which can be understood by the internal controller of the drone. The control command will be turned into a MAVLink message, which can be interpreted by the flight control unit (FCU) of the drone to control the drone directly.

Deep neural networks in the action selector were implemented based on a version of Google's Inception, which achieved the state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC2014) [14]. Thus, team UNITS recorded a large number of video clips showing how human control a drone to fly through a square hoop. The video clips were recorded from the perspective of the drone. Some frames in the video clips were labeled with human-generated control signals which are vectors of integers, each of them denotes an action such as turning left, turning right, and staying put. Currently, each neural network can output three different integer values only.

As shown in Fig. 7, a drone starts at an initial position and aims to fly through a hoop. In the two-step procedure, the drone first flies to the center line, which is a normal vector of the 2D plane of the hoop, and then flies towards the hoop along the center line through the center point of the hoop. The success of this maneuver depends on whether the drone can carry out these two steps correctly and quickly.

Experiments carried out compare the performance of the drone in terms of the success rate, the distance from the center line, and the time to fly to the center line, using two different action selectors trained with different sizes of the training sets. It is expected that an action selector whose DNNs are trained using a larger training set will outperform the one using a smaller training set.

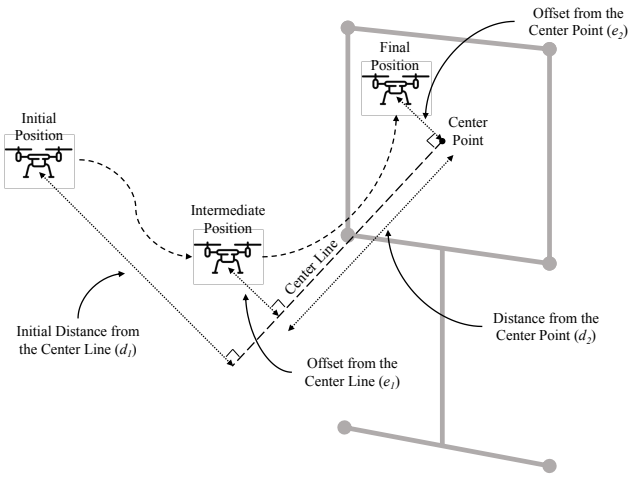


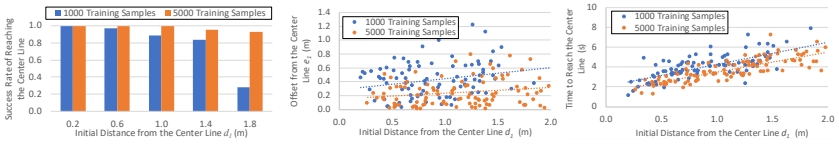
Fig. 7 The two-step procedure for flying through a hoop using the action selector developed by Team UNIST.

The experimental setup is shown in Fig. 7. First of all, an initial position of the drone in front of the hoop is randomly picked. The initial position should be at least 2 meters from the hoop and the camera on the drone should be able to see the hoop at the initial position so that the action selector can select the actions based on the images in the video stream. Then the action selector

will generate control commands to control the drone to fly towards the center line based on what it sees about the hoop. Each DNN in the action selector can generate three different values. When these values are zero, it means that the drone has reached the setpoint (i.e., the distance between the drone and the center line is small enough), and therefore the drone should stabilize at the current position. However, there are situations in which the drone fails to stabilize, either 1) it flies around the center line and cannot stabilize for a long time, or 2) it flies away from the center line and never comes back. Fig. 8(a) shows the success rate of stabilization near the center line. As can be seen, when the initial distance from the center line (d_1) increases, the success rate decreases. However, if the size of the training set is too small (e.g., 1000), the success rate drops rapidly; by contrast, the success rate can always maintain over 90% when the size of the training set is 5000. Hence, the performance of DNNs plays an important role in stabilizing the drone near the center line.

Among the cases in which the drone can stabilize, the distance between the drone and the center line is measured. As shown in Fig. 7, suppose that a drone stabilizes at a position called the intermediate position, the distance between the drone and the center line is the offset e_1 between the intermediate position and the center line. Fig. 8(b) shows the offsets from the center line as the initial distance d_1 increases. Notice that there are a lot more orange dots than blue dots because the success rate of stabilization is much higher when the size of the training set is 5000. As can be seen, the offset is much smaller when the training set is large. In general, the offset slightly increases as the initial distance increases, perhaps due to the fact that outliers occur more often when d_1 is large. Hence, the performance of DNNs can greatly affect how close the drone can stabilize near the center line.

Finally, the time the drone took to stabilize near the center line was also measured. Fig. 8(c) shows the time to reach the center line as the initial distance d_1 increases. As expected, the time increases linearly with d_1. However, a larger training set can help to stabilize more quickly. This experiment shows that the performance of the drone can be improved by increasing the number of training data.



(a) (b) (c)

Fig. 8 The performance of UNIST's drone when flying towards the center line. (a) The initial distance from the center line (d_1) versus the success rate of reaching the center line. (b) The initial distance from the center line (d_1) versus the offsets from the center line (e_1). (c) The initial distance from the center line (d_1) versus the time to reach the center line.

8 Conclusion

This paper introduced the aerial vehicles and approaches used in the two ADR competitions in conjunction with IROS 2016 and IROS 2017 that both intended to test autonomous drone navigation for a known cluttered environment. Successful teams implemented waypoint tracking methods along with robust gate recognition algorithms. Although autonomous drones tend to suffer from large position errors as they traverse the arena, accurate tracking leads to successful flight through closed gates in most occasions. The difficulties in completing the racing track, however, still remain a challenge. In the future ADR competitions, the amount of environmental information available to drones will be reduced, aiming at pushing for more autonomy in drone technologies.

Acknowledgment

J. Martinez-Carranza is thankful for the funding received by the Royal Society through the Newton Advanced Fellowship with reference NA140454. Team UZH thanks Elia Kaufmann, Antoni Rosinol Vidal, and Henri Rebecq for their great help in the software implementation and integration. Team of TU Delft would like to thank the organizers of the Autonomous Drone Race event. Team UNIST's work was supported by NRF (2.180186.01 and 2.170511.01) All authors would like to thank the organizers of the Autonomous Drone Racing.

References

1. H. Moon, Y. Sun, J. Baltes, and S. J. Kim, "The IROS 2016 competitions," *IEEE Robotics and Automation Magazine*, vol. 24, no. 1, pp. 20–29, 2017.
2. P. Brisset, A. Drouin, M. Gorraz, P.-S. Huard, and J. Tyler, "The paparazzi solution," in *2nd US-European Competition and Workshop on Micro Air Vehicles (MAV)*, 2006.
3. R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.
4. L. O. Rojas-Perez and J. Martinez-Carranza, "Metric monocular SLAM and colour segmentation for multiple obstacle avoidance in autonomous flight," in *IEEE 4th Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, October 2017.
5. "Object modelling by registration of multiple range images," *Image and Vision Computing*, vol. 10, no. 3, pp. 145–155, 1992.
6. J. D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*. Addison-Wesley Longman Publishing Co., Inc., 1982.
7. M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, 2018.
8. G. de Croon, M. Perçin, B. Remes, R. Ruijsink, and C. De Wagter, *The DelFly: Design, aerodynamics, and artificial intelligence of a flapping wing robot*. Springer, 2016.
9. S. Jung, S. Cho, D. Lee, H. Lee, and D. H. Shim, "A direct visual servoing-based framework for the 2016 IROS Autonomous Drone Racing Challenge," *Journal of Field Robotics*, vol. 35, no. 1, pp. 146–166, 2017.

10. S. Jung, S. Hwang, H. Shin, and D. H. Shim, "Perception, guidance and navigation for indoor autonomous drone racing using deep learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2539–2544, 2018.
11. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *European conference on computer vision (ECCV)*, pp. 21–37, Springer, 2016.
12. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.
13. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105, 2012.
14. C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, Inception-ResNet and the impact of residual connections on learning," in *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 4278–4284, 2017.