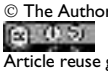
Check for updates

Original Article

Structural Health Monitoring
2024, Vol. 23(2) 971–990
© The Author(s) 2023

Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/14759217231177314
journals.sagepub.com/home/shm


Deep learning-based obstacle-avoiding autonomous UAVs with fiducial marker-based localization for structural health monitoring

Ali Waqas[1], Dongho Kang[2] and Young-Jin Cha[1]

Abstract

This paper proposes a framework for obstacle-avoiding autonomous unmanned aerial vehicle (UAV) systems with a new obstacle avoidance method (OAM) and localization method for autonomous UAVs for structural health monitoring (SHM) in GPS-denied areas. There are high possibilities of obstacles in the planned trajectory of autonomous UAVs used for monitoring purposes. A traditional UAV localization method with an ultrasonic beacon is limited to the scope of the monitoring and vulnerable to both depleted battery and environmental electromagnetic fields. To overcome these critical problems, a deep learning-based OAM with the integration of You Only Look Once version 3 (YOLOv3) and a fiducial marker-based UAV localization method are proposed. These new obstacle avoidance and localization methods are integrated with a real-time damage segmentation method as an autonomous UAV system for SHM. In indoor testing and outdoor tests in a large parking structure, the proposed methods showed superior performances in obstacle avoidance and UAV localization compared to traditional approaches.

Keywords

Autonomous flight, unmanned aerial system, damage segmentation, obstacle avoidance algorithm, UAV localization, deep learning, GPS-denied areas

Introduction

Deep learning-based structural damage detection is a hot topic in the construction and structural health monitoring (SHM) field. Cha et al.[1,2] proposed a new deep convolutional neural network (DCNN) for structural damage detection using vision sensors. It has received enormous interest worldwide, with numerous studies following.[3–5] To quantify detected damage, damage segmentation methods using deep learning have been extensively investigated by many researchers. For example, U-Net[6] and its transformed algorithm have been applied.[7–9] Choi and Cha[10] suggested a fast, efficient deep semantic network for crack segmentation in complex scenes by using an atrous pyramid pooling network and "DenSep" modules. Saleem et al.[11] applied Mask R-CNN[12] for crack segmentation. They tested image data of a real bridge structure collected by unmanned aerial vehicles (UAVs). Yu et al.[13] developed a DCNN and enhanced chicken swarm algorithm to improve the accuracy and computational cost for the automated detection of concrete cracks. Abdeljaber et al.[14] developed a novel, fast, and accurate real-time structural damage detection system using one-dimensional convolutional neural networks (1D CNNs) that automatically extract damage-sensitive features from raw acceleration signals. In another study, a new approach for detecting compressive stress and load-induced damage in concrete structures using a DCNN integrated with electromechanical admittance (EMA) was proposed.[15] Two-dimensional convolutional neural network (2D CNN) for rapid damage quantification in concrete structures using raw EMA data has also been used.[16] Potenza et al.[17] suggested color histogram-based segmentation for UAV data collected from a real bridge structure. Kang and Cha[18]

[1]Department of Civil Engineering, University of Manitoba, Winnipeg, MB, Canada
[2]Ericsson, Toronto, ON, Canada

Corresponding author:
Young-Jin Cha, Department of Civil Engineering, University of Manitoba, SP427 Stanley Pauley Engineering, 105 Dafoe Rd W, Winnipeg, MB R3T 6B3, Canada.
Email: young.cha@umanitoba.ca

developed a semantic transformer representation network (STRNet) to segment cracks at the pixel level in a complex background scene. It shows state-of-the-art performance in terms of accuracy and processing speed. The reported mean intersection over union (mIoU) is 92.6%, with 49.2 frames per second (FPS) and an image size of $1024 \times 512 \times 3$. However, the method has not yet been implemented with UAV images but rather with images collected from fixed-installed cameras. Bao and Li[19] proposed that machine learning can be used to analyze monitoring data to discover and model the performance and conditions of a structure, thereby creating a "machine learning paradigm" for SHM.[19] In another study, a fusion CNN architecture was developed for crack identification in the steel box girders of bridges based on real-world images containing complicated disturbance information.[20]

UAVs are attractive for various applications, including SHM, since they can overcome accessibility difficulties without heavy machinery, reduce overall costs, and increase the frequency of inspections.[21,22] For example, McGuire et al.[22] tested a commercial UAV for bridge and tower inspections. Benz et al.[23] conducted a bridge inspection using a UAV and suggested a new crack segmentation algorithm based on a pretrained CNN. However, all these applications are limited to manual flights. Manually flying a UAV requires a skillful pilot and thus is costly, nor is it feasible for mass application to numerous bridge structures.

Recently, Kang and Cha[24] and Ali et al.[25] developed an autonomous UAV flight method for structural damage detection and localization in areas where a global position system (GPS) signal is not available. Generally, GPS signals are weak or unavailable beneath a bridge deck or indoors. To replace the GPS signal, an ultrasonic beacon system (USB) was used to generate a three-dimensional (3D) pseudo map. Additionally, a deep CNN[25] was used to detect cracks on concrete floors. Kang and Cha[24] study was the first trial of an autonomous UAV system for SHM. However, USB signals are vulnerable to various magnetic fields due to transmission towers or electric vehicles and machinery. Further, the method did not have a function to avoid obstacles in the planned trajectory of autonomous navigation.

Obstacles such as beams or pillars can present difficulties for this UAV-based SHM. There are various solutions to avoid obstacles. For example, González et al.[26] implemented a one-degree laser distance sensor in the frontal direction of a UAV. In their research, the UAV detected a wall by using the distance sensor, but this 1D sensor missed small obstacles or obstacles that are not in field of the (1D) sensor. Song et al.[27] implemented a 3D light detection and ranging (LIDAR)-based simultaneous localization and mapping for obstacle avoidance and 3D mapping for bridge inspection. 3D mapping using 3D LIDAR is promising for future SHM, but 3D LIDAR is much more expensive and much heavier than an RGB camera. Therefore, it may require very large payload UAVs, which are not appropriate for navigating inter-story or narrow areas, such as between pillars or bridge piers.

Although in the SHM field, research on obstacle avoidance methods (OAMs) for UAVs has not yet started, obstacle avoidance for UAVs is a popular topic in robotics. For example, Mori and Scherer[28] suggested one of the first attempts for a single camera-based obstacle avoidance algorithm for a UAV. The algorithm detects the obstacle by using speeded-up robust features[29] and avoids it. More recently, Smolyanskiy et al.[30] and Loquercio[31] suggested deep learning-based OAMs by supervised training of a CNN to predict steering angle and collision probability in a road scene image for UAV control, but this approach is not guaranteed to respond to unexpected environmental features because the algorithm is limited to the variety of training data. Afterward, depth estimation from a single camera was developed for obstacle distance estimation.[32,33] However, none of these approaches have been implemented in the SHM field.

In this paper, to overcome the limitations of the USB and realize an obstacle avoidance autonomous UAV system for SHM in a GPS-denied environment, a new OAM based on fiducial markers is proposed. Fiducial markers do not need electric batteries to operate, and small markers (4 cm $\times$ 4 cm) can be attached at UAV waypoints to establish a navigation trajectory. The developed autonomous UAV system was tested in indoor and outdoor environments, and the collected videos were analyzed by STRNet to segment cracks at the pixel level. The paper is organized into three additional sections. Section "Methodology" describes the proposed OAM with a fiducial marker-based autonomous navigation method and provides a brief explanation of STRNet. In Section "Case studies," the experimental tests and performances of the OAM and STRNet are discussed. Section "Conclusion and future direction" presents the conclusion and limitations and suggests future improvements.

Methodology

An obstacle-avoiding autonomous UAV navigation control method is proposed for civil SHM. The automated navigation method is composed of a two-level fight controller using two proportional -integral -derivative (PID) controllers and an object avoidance method to modify the planned trajectory of the UAV system when the UAV encounters obstacles, as shown

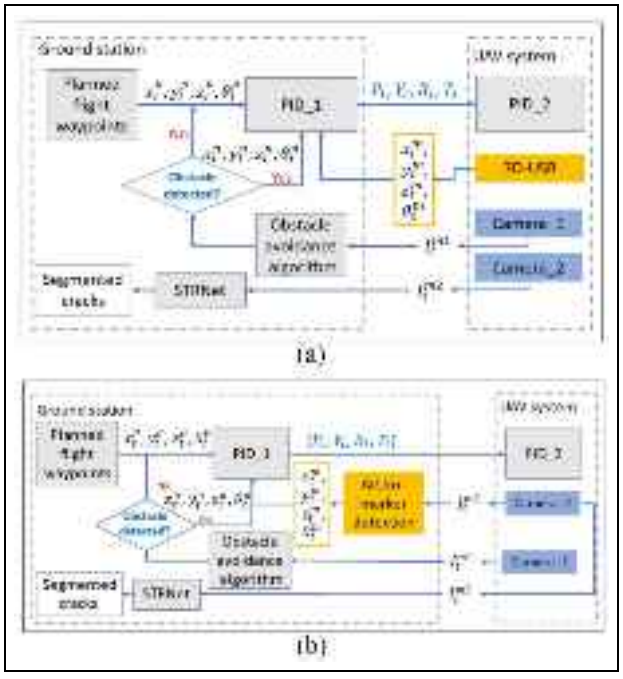


Figure 1. Autonomous UAV-based SHM system architecture. (a) 3D-USB-based autonomous navigation and (b) ArUco marker-based autonomous navigation.
SHM: structural health monitoring; UAV: unmanned aerial vehicle; USB: ultrasonic beacon system.

in Figure 1. We propose two types of applications for the OAM, namely 3D USB-based (https://marvelmind.com/) and fiducial ArUco marker-based[34] (https://www.uco.es/investiga/grupos/ava/portfolio/aruco/.) localization techniques to realize autonomous navigation in GPS-denied areas (Figure 1).

To inspect civil infrastructure, as the first step, waypoints of the UAV navigation should be defined using either 3D USBs or ArUco markers. The *planned* waypoints are a series of 3D coordinates composed of x_t^p, y_t^p, z_t^p, and θ_t^p, which stand for ground longitudinal coordinate, transverse coordinate, vertical (height) coordinate, and direction of UAV (yaw), respectively. p and t stand for planned waypoints and time steps during the navigation, respectively. The first PID controller (PID_1) calculates the required pitch (P_t), yaw (Y_t), roll (R_t), and throttle (T_t) to achieve the planned waypoints $(x_t^p,\ y_t^p, z_t^p, \theta_t^p)$ based on the discrepancy with the current *measured* coordinates $(x_t^m,\ y_t^m, z_t^m, \theta_t^m)$. In the proposed OAM, if an obstacle is detected, the OAM generates a new waypoint (i.e., $x_t^n,\ y_t^n, z_t^n, \theta_t^n$) in each time step to fully avoid it.

For the first approach (Figure 1(a)), a 3D pseudo map is constructed using four stationary beacons in the area the UAV will navigate. The measured coordinates $(x_t^m,\ y_t^m, z_t^m)$ indicate the actual location of the mobile beacon mounted on the UAV, which is localized by calculation of the 3D distances compared to the four stationary beacons. The yaw angle, θ_t^m, is calculated by the magnetic compass sensor inside the UAV. The calculated P_t, Y_t, R_t, and T_t are inputted to PID_2, which is installed in the UAV to generate a current for each motor of the wings of the UAV. Two cameras are installed on the UAV: one (Camera_1) for front monitoring and the other (Camera_2) for downward floor inspection. The video images, I_t^{m1}, from Camera_1 are processed by an obstacle avoidance algorithm, and if there is an obstacle, a new waypoint $\left(x_t^n,\ y_t^n, z_t^n, \theta_t^n\right)$ is determined to avoid collision. The other video images, I_t^{m2}, from Camera_2 are analyzed through a pixel-level segmentation method (i.e., STRNet).

For the second approach (Figure 1(b)) of localization, ArUco markers are used to create a 3D-pseudo map in the area the UAV will navigate. This method detects markers in the incoming image frame I_t^{m2} and decrypts them to find the imbedded binary code of each marker in the marker bank, which has all the information of each marker's number and coordinates in the pseudo map. Using these decrypted marker numbers and coordinates, the UAV's position in the 3D-pseudo map $\left(x_t^m, y_t^m, z_t^m\right)$ and current yaw θ_t^m are calculated for UAV localization (See Section "ArUco marker detection"). These measured values are given as input to PID_1, which calculates the desired control values P_t, Y_t, R_t, and T_t. More details of the obstacle avoidance algorithm, ArUco marker detection module, UAV localization, and STRNet are provided in the subsections.

Obstacle avoidance method

The overall procedure of the obstacle avoidance algorithm that we propose in this paper (as shown in Figure 2) is as follows: (1) obstacles are detected by You Only Look Once version 3 (YOLOv3)[35] using the input image, I_t^{m1}, in Figure 1; (2) the YOLOv3 (https://pjreddie.com/darknet/yolo/) provides the bounding boxes to localize the obstacles within the image; and (3) if there are multiple bounding boxes for multiple obstacles, box clustering should be done first using the standard deviation (SD) filter and K-means clustering method. This box clustering provides multiple box groups, and decision-making should be done to avoid the closer obstacle.

In this study, the UAV can avoid the obstacle by moving either of right or left direction. The decision is made based on the location of the center point of the clustered bounding box compared to the center point of the entire image, I_t^{m1}. Each step is detailed below.

Detection of obstacle using YOLOv3. To detect any number of obstacles, YOLOv3, which is composed of an

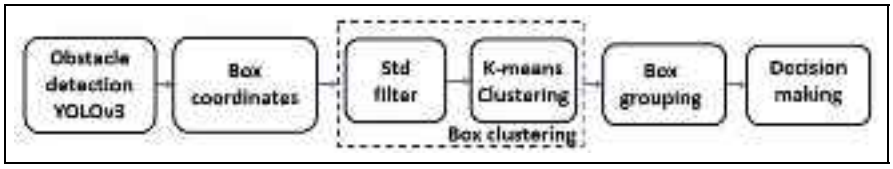


Figure 2. Steps in obstacle avoidance.

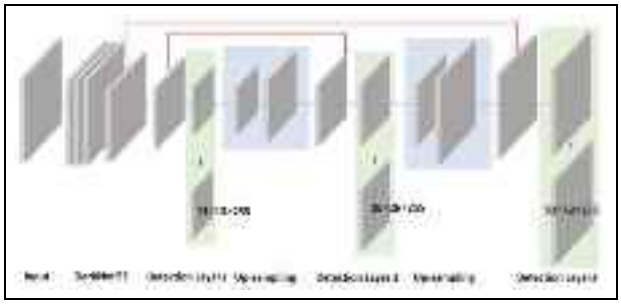


Figure 3. YOLOv3 architecture.

Figure 4. Obstacle detection process.

encoder and decoder, is used as an example network. It can be replaced by any recent version of YOLO series. DarkNet53[36] is used as the encoder. The decoder of the YOLOv3 consists of three detection layers (expressed as greenish boxes in Figure 3) and upsampling layers having different scales (expressed as blueish boxes in Figure 3).

The input size of YOLOv3 is resized to $416 \times 416 \times 3$ from the original image size of $540 \times 480 \times 3$. DarkNet 53 is used to extract the feature maps having sizes of $13 \times 13 \times 255$, $26 \times 26 \times 255$, $52 \times 52 \times 255$ in "Detection Layer 1," "Detection Layer 2," and "Detection Layer 3," respectively, as shown in Figure 3. The main purpose of using these multiple feature maps by upsampling is to consider different levels of features to improve the object (obstacle) detection performance. The depth of the feature map is evenly 255, composed of three anchors (i.e., anchor 1, anchor 2, and anchor 3). Each anchor dimension is $1 \times 1 \times 85$ to represent bounding box coordinates $\left(b_x, b_y, b_w, b_h\right)$, object confidence, and class scores $(C_1, \cdots, C_n)$, as shown in Figure 4.

As an example, multiple chairs are used as obstacles. The bounding box coordinates, object confidence, and class scores are multiplied by the sigmoid function, and the bounding box width, b_w, and height, b_h, are multiplied by nine predefined box ratios, which can be found in the original paper.[35] The nonmaximum suppression algorithm is applied to remove the overlapped same-class object. When there are two bounding boxes having the same class, and the boxes are overlapped by more than 50%, the bounding box with a lower-class score is deleted.

Obstacle clustering. YOLOv3 provides N number of bounding boxes depending on the number of obstacles in the input image, where $N = \{1, 2, 3 \cdots n\}$. The resulting bounding box coordinates $[(b_x^1, b_y^1) \cdots (b_x^n, b_y^n)]$ with dimensions $\left[\left(b_w^1, b_h^1\right) \cdots \left(b_w^n, b_h^n\right)\right]$ are inputted to the obstacle clustering method that we propose in this paper, as shown in Figures 2 and 5.

As the first step of the method, a "Size filter" (i.e., $b_w * b_h > \alpha$) is proposed to remove extremely small boxes, which can be assumed to indicate that the object is too small or too far from the UAV. The next step is to determine the distribution level of the obstacles. An "standard deviation (STD) filter" is formulated as Equation (1).

$$\text{std}\left[\left(b_y^1 + \frac{b_h^1}{2}\right) \cdots \left(b_y^n + \frac{b_h^n}{2}\right)\right] > \beta \qquad (1)$$

where $b_y^n + \frac{b_h^n}{2}$ is the size index of the object and the distance of the object from the UAV. When the calculated STD value is smaller than the threshold (β), it is assumed that there are multiple obstacles as a single group; then, without processing of the K-means clustering, we do "Box grouping," as shown in Figure 5. The threshold values, α, and β, are defined as 1500 and 20, respectively; these can be determined by the user's engineering judgment by considering the flight environment and possible obstacle sizes. The range of α can be set between 0 and the total number of pixels in the image. During the YOLOv3 testing for different obstacle configurations, we observed that all significant obstacles were represented by a bounding box, where

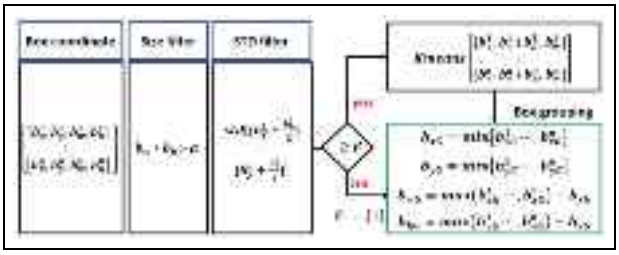


Figure 5. Box clustering algorithm.

$b_w * b_h > 1500$. Therefore, smaller bounding boxes were noise or nonsignificant obstacles to improve processing speed in the real-time obstacle avoidance algorithm. Similarly, the β hyperparameter was set based on the testing results of images using YOLOv3. We observed using trial and error that 1500 and 20 were optimal values for α and β, respectively, to avoid obstacles and maintain efficient and safe real-time avoidance of obstacles.

When the calculated STD value is greater than β, K-means clustering is applied to form multiple groups. This K-means clustering generates multiple groups to the bounding boxes that are provided by the YOLOv3, as shown in Figure 6.

The output from K-means is $[[b^1_{xG}, b^1_{yG}, b^1_{wG}, b^1_{hG}] \cdots [b^n_{xG}, b^n_{yG}, b^n_{wG}, b^n_{hG}]]$, where $G = 1, 2, \ldots, g$ is the assigned group of the bounding boxes. Finally, the bounding box size and coordinates of each group are determined by the box grouping formulation expressed in Equation (2).

$$\begin{aligned} \dot{b}_{xG} &= \min\left[b^1_{xG} \cdots b^n_{xG}\right] \\ \dot{b}_{yG} &= \min[b^1_{yG} \cdots b^n_{yG}] \\ \dot{b}_{wG} &= \max\left(b^1_{xG}, \cdots, b^n_{xG}\right) - \dot{b}_{xG} \\ \dot{b}_{hG} &= \max(b^1_{yG}, \cdots, b^n_{yG}) - \dot{b}_{yG}. \end{aligned} \tag{2}$$

Using the calculated values $(\dot{b}_{xG}, \dot{b}_{yG}, \dot{b}_{wG}, \dot{b}_{hG})$ from Equation (2), we can classify the obstacles into distinct groups (red- or blue-colored outlines of bounding boxes are shown in Figure 6(b))

Estimation of distance of UAV from obstacle. Now, we should calculate the distance between the UAV camera and the obstacle to figure out how to avoid it. For estimation of the distance of the UAV from the obstacle, it is assumed that width W and height H of the obstacle are known because YOLOv3 is used to predict the bounding box and class of obstacle. To increase the robustness of our algorithm and adapt it to different types of obstacles, we logged the heights and widths of various obstacles that may be present in the SHM

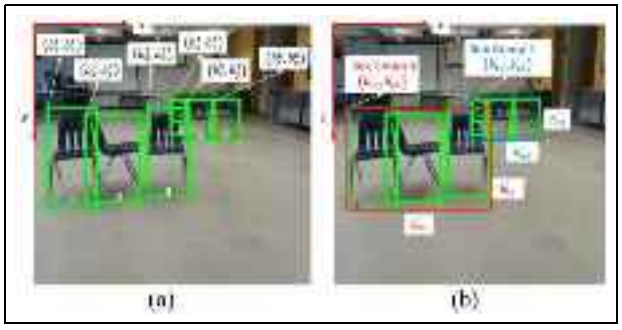


Figure 6. Obstacle detection and obstacle clustering: (a) original YOLOv3 result and (b) obstacle clustering result.

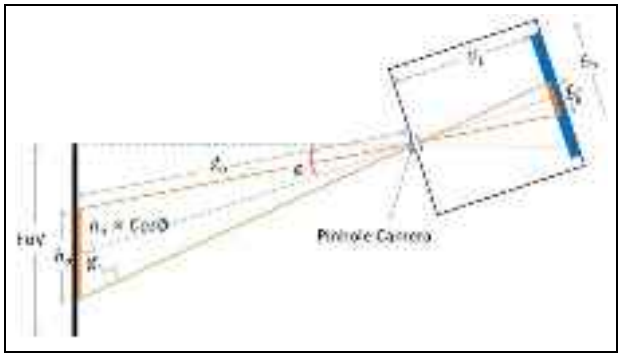


Figure 7. Schematic view of a pinhole camera model.

environment. To obtain the distance of the object from the camera, the pinhole camera model is used, as shown in Figure 7.

In the first step of distance calculation, the obstacle is detected in image I^{m2}_t (see Figure 1) in the form of a bounding box with dimensions $(\dot{b}_{xg}, \dot{b}_{yg}, \dot{b}_{wg}, \dot{b}_{hg})$ as shown in Figure 6(b). For distance calculation, first, the projection of the obstacle's bounding box on the camera sensor is calculated, as shown in Equations (3) and (4).

$$\dot{b}^s_h(\text{mm}) = \frac{\dot{b}_h(px)}{H(px)} \times S_h(\text{mm}), \tag{3}$$

$$\dot{b}^s_w(\text{mm}) = \frac{\dot{b}_w(px)}{W(px)} \times S_w(\text{mm}), \tag{4}$$

where $\dot{b}^s_h$ and $\dot{b}^s_w$ are the height and width of the bounding box projection on the camera sensor in mm, respectively. S_h and S_w are the height and width of the camera sensor provided in the technical specifications of the camera, respectively. Moreover, using Zhang's calibration technique,[37] the camera's intrinsic parameters including focal length F_l are calculated. Using geometric equalities from Figure 7, d_o can be calculated using Equation (5)[38]:

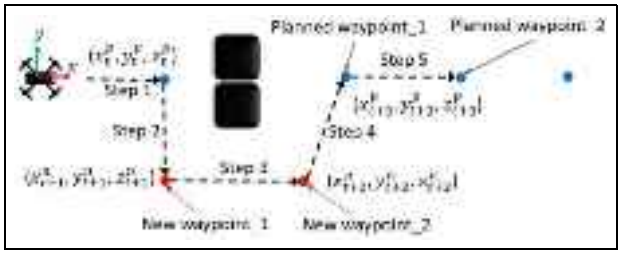


Figure 8. Generating new waypoints with OAM.
OAM: obstacle avoidance method.

$$d_o = \frac{h_o}{\cos(\theta)} \times \frac{F_l(\text{mm})}{\dot{b}_h^s(\text{mm})} \tag{5}$$

where θ is the tilt of camera with respect to the horizontal plane and $h_o \cos(\theta)$ is the plan of the object perpendicular to the camera sensor plan, as shown in Figure 7.

Generating new waypoint. When obstacles are detected in Step 1 in Figure 8, the proposed OAM has to figure out a new waypoint based on the distance between the UAV and the obstacles. In this case, only the first group of obstacles closer to the UAV (shown in Figure 6(b) as a red rectangle) is considered. When the calculated distance d_o between the nearest obstacle and the UAV is less than a threshold distance, the OAM assumes that the obstacle is close enough to the UAV, and the UAV is stopped automatically at its current location to avoid the collision. Through experiments, the threshold distance value can be adjusted between 1 m and 2 m depending on safety concerns.

The next step is to determine a new waypoint to avoid obstacles. To do this, the center $\left(\frac{W}{2}, \frac{H}{2}\right)$ of the image I_t^{m2} and the center $(\hat{x}, \hat{y})$ of the bounding box are compared. If $\left(\frac{W}{2} - \hat{x}\right)$ is positive, as shown in Figure 8, the new waypoint is generated at the right side with a 1 m distance, that is, $\left(x_t^n,\ y_t^n - 1m,\ z_t^n\right) = \left(x_{t+1}^n, y_{t+1}^n, z_{t+1}^n\right)$ in Step 2, as shown in Figure 8. Here, $\frac{W}{2}$ is the horizontal center of the field of view. This rule can be expressed as shown in Equation (6)

$$x_{t+1}^n,\ y_{t+1}^n,\ z_{t+1}^n = \begin{cases} \left(\frac{W}{2} - \hat{x}\right) > 0 & : x_t^n,\ y_t^n - 1m,\ z_t^n \\ \left(\frac{W}{2} - \hat{x}\right) \leqslant 0 & : x_t^n,\ y_t^n + 1m, z_t^n \end{cases} \tag{6}$$

If $\left[\frac{W}{2} - (\dot{b}_x + \dot{b}_w)\right] \leqslant 0$, the UAV should be moved by one more meter to the right (along y axis). This should be continued until $\left[\frac{W}{2} - (\dot{b}_x + \dot{b}_w)\right] > 0$. If the new waypoint is sufficient $\left[\frac{W}{2} - (\dot{b}_x + \dot{b}_w)\right] > 0$ to avoid the obstacles, then another waypoint $\left(x_{t+2}^n,\ y_{t+2}^n,\ z_{t+2}^n\right)$ is generated $= \left(x_{t+1}^n + 1m,\ y_{t+1}^n,\ z_{t+1}^n\right)$ as Step 3 in Figure 8. Equation (7) describes the two cases of decisions for Step 3, in which the UAV avoids the left or right obstacles from Step 2

$$\begin{aligned} &x_{t+2}^n,\ y_{t+2}^n,\ z_{t+2}^n = \text{Left} \\ &\begin{cases} \frac{W}{2} - \dot{b}_x < 0 & : x_{t+1}^n + 1m,\ y_{t+1}^n,\ z_{t+1}^n \\ \frac{W}{2} - \dot{b}_x \geqslant 0 & : x_{t+1}^n + 1m,\ y_{t+1}^n + xm,\ z_{t+1}^n \end{cases} \\ &x_{t+2}^n,\ y_{t+2}^n,\ z_{t+2}^n = \text{Right} \\ &\begin{cases} \frac{W}{2} - (\dot{b}_x + \dot{b}_w) < 0 & : x_{t+1}^n + 1m,\ y_{t+1}^n - xm,\ z_{t+1}^n \\ \frac{W}{2} - (\dot{b}_x + \dot{b}_w) \geqslant 0 & : x_{t+1}^n + 1m,\ y_{t+1}^n,\ z_{t+1}^n \end{cases} \end{aligned} \tag{7}$$

At the new waypoint $\left(x_{t+2}^n,\ y_{t+2}^n,\ z_{t+2}^n\right)$, the obstacles are disappeared from the image frame; then, the UAV moves to the originally planned waypoint $\left(x_{t+3}^p,\ y_{t+3}^p,\ z_{t+3}^p\right)$ to return to the originally planned trajectory. All these processes are repeated until the end of the planned navigation.

ArUco marker detection

To navigate a GPS-denied area autonomously, a USB was implemented by Kang and Cha.[24] However, the USB was vulnerable to magnetic interference; therefore, sometimes, it did not provide reliable signals and resulted in poor localization of the UAV, which is explained in later sections. To overcome this limitation of the USB, we introduce an ArUco marker-based UAV localization method.[34,39] An ArUco marker is a square-shaped marker, as shown in Figure 9, with a black and white square patch containing a unique barcode. Multiple numbers of installed ArUco makers establish a 3D pseudo map to replace GPS signals, and each ArUco marker's 3D coordinates based on this 3D pseudo map are saved in a marker library in the ground station. Then, we can set the starting waypoint of the UAV based on this 3D map. Because each marker has a unique binary barcode, the barcode number is used to find its location information in the library; therefore, we must read the barcode using image processing.

The first step is to detect this barcode from I_t^{m2}, as shown in Figure 1. For this purpose, the image $\left(I_t^{m2}\right)$ is converted to grayscale to extract the barcode, as shown in Figure 9(a). The grayscale image is binarized using local adaptive thresholding to extract the edges of the object in the frame, as shown in Figure 9(b). After that, contours are extracted from the binarized image by using Suzuki and Abe's algorithm,[40] which produces many contours, most of which are irrelevant to marker detection. We only need rectangular contours for marker detection because of the marker's shape; therefore, using the Douglas–Peucker algorithm,[41] polygonal approximation is performed, and contours with four corners are selected for further processing. Then, a

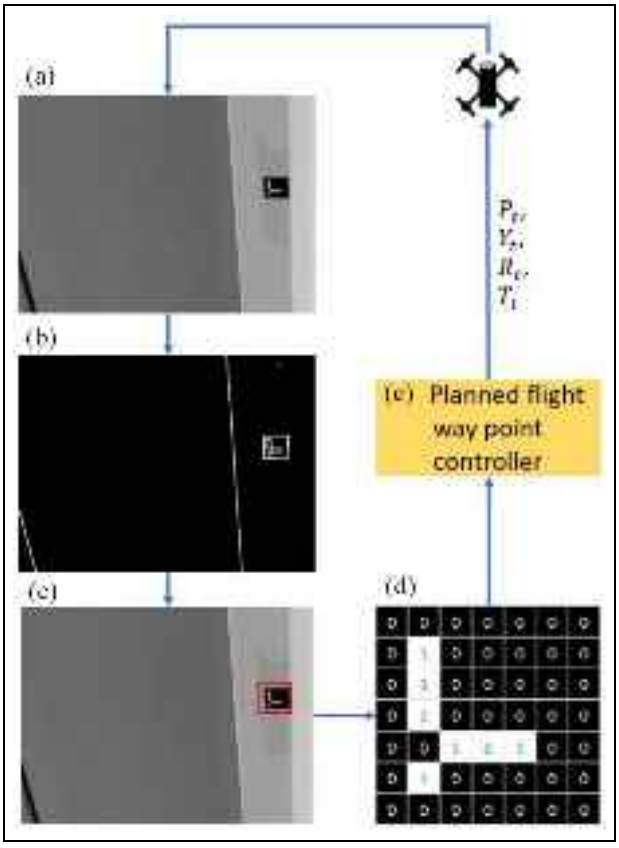


Figure 9. Schematic block diagram of ArUco marker detection process: (a) convert image to grayscale, (b) local adaptive thresholding and contour filtering, (c) rectangular image patch selection, and (d) assign the binary value 1 or 0.

valid rectangular contour patch is extracted from the grayscale image, as shown in Figure 9(c). The extracted rectangular image patch should be converted to a true black-and-white image by using Otsu's method.[42] Then, the rectangular image patch is divided into equally sized 7×7 grids, as shown in Figure 9(d). For all black grids, 0 value is assigned, and for all white grids, 1 value is assigned (see Figure 9(d)). Using these 7×7 grids, only the core 5×5 grids are used to read the barcode of the marker to determine the number (ID) of the marker. The marker's ID in the marker library is used to find the coordinates of the detected marker. So, each marker's coordinates should be predefined based on the location of each marker within the 3D pseudo map. Using the coordinates of the detected marker, the current location of the UAV can be calculated within the 3D pseudo map, as is explained in detail in the next section.

The ArUco marker detection method is robust against environmental distractions by using unique corner detection and increasing the number of bits in the marker. Using markers with a higher number of bits ensures that the pattern is complicated, so that it does not occur in the monitoring environment and improves robustness. Additionally, false detections can be filtered by using a predefined dictionary of markers. If a fake marker is detected and it exists in the dictionary, it will not affect the UAV's position as it corrects its position based on the next true marker detected.

Figure 10. Attached ArUco markers (white circle) in the ceiling of the reinforced concrete structure.

UAV localization. For autonomous flight using only ArUco markers, UAV spatial coordinates and pose $(x_t^m,\ y_t^m, z_t^m,$ and $\theta_t^m)$ in the pseudo map must be calculated from the incoming image I_t^{m2}. For this localization, the first step is to create a 3D pseudo map by installing markers, which are the waypoints in the flight zone. Figure 10 shows an example of attached ArUco markers (depicted as white circles) as the waypoints for the UAV to generate a planned trajectory of the flight.

Each marker in the pseudo map has its own local coordinate system $\boldsymbol{O}_A(x_{Ai},\ y_{Ai},\ z_{Ai})$ having an origin at the center of the marker where i stands for the ith marker in the pseudo map, as shown in Figure 11. Similarly, a UAV has its own local coordinate system $\boldsymbol{O}_d(x_d, y_d, z_d)$. The centroidal axis of the UAV body is x_d, horizontal direction is y_d, and vertical direction is z_d. The camera coordinate system $\boldsymbol{O}_c(x_c, y_c, z_c)$ has its origin at the optical center of the camera, with the x-axis pointing right and the y-axis pointing front, as shown in Figure 11. The pixel coordinate system of the marker image is given by $\boldsymbol{O}_f\left(u_f, v_f\right)$ with the origin at the top left of the image frame I_t^{m2}.

In the first step of localization, the marker's unique barcode information is decrypted from the incoming frame I_t^{m2} as discussed in Section "ArUco marker detection." The marker library contains four possible

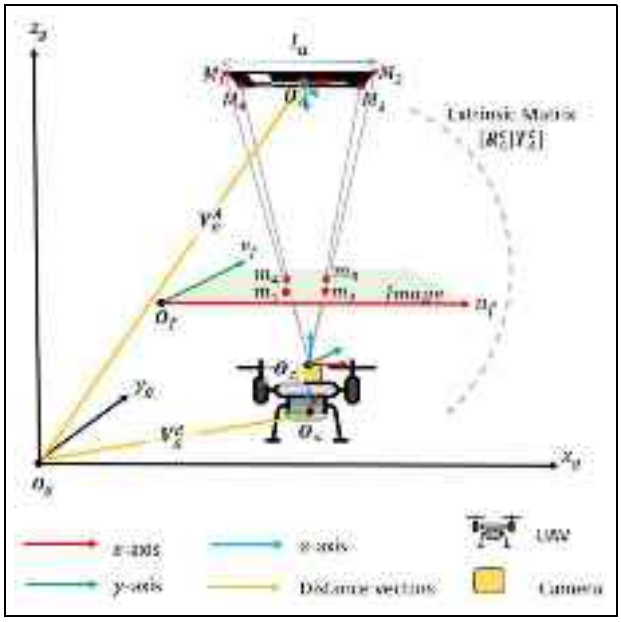


Figure 11. UAV localization in the pseudo map using ArUco markers.
UAV: unmanned aerial vehicle.

orientations of a barcode for each marker, which are used to recognize the corners $\boldsymbol{M_p} = (M_1, M_2, M_3, \text{and } M_4)$ where $\boldsymbol{p} = 1, 2, 3, 4$ and assign local coordinates $\boldsymbol{O}_A = (x_{Ai}, y_{Ai}, z_{Ai})$ to the marker, as shown in Figure 11.

Next, to find the location and pose of the camera using ArUco markers, we adopted Burger's pose of the camera estimation method[43] to this UAV camera pose estimation problem. It is assumed that the markers' dimensions $(l_A \times l_A)$ cm in real-world units are known and the marker is on a plane surface. The coordinates of each corner point $M_p = (X_p, Y_p, 0)^T$ of the $\boldsymbol{O}_A$, ArUco coordinate system and its two-dimensional projection in pixel coordinates $m_p = (u_p, v_p)^T$ are also known. $\bar{M}_p = (\bar{X}_p, \bar{Y}_p, 0, 1)^T$ and $\bar{m}_p = (\bar{u}_p, \bar{v}_p, 1)^T$ are the projective coordinates of M_p and m_p, respectively, for convenience of mathematical calculation. The aim of UAV localization is to find the extrinsic parameters of the UAV camera $[\boldsymbol{R}_A^c | \boldsymbol{T}_A^c]$, which is a 3×4 matrix as shown in Equation (8), where $\boldsymbol{R}_A^c$ and $\boldsymbol{T}_A^c$ are rotation and translation transformation of the ArUco marker compared to the camera coordinate system $\boldsymbol{O}_c$, respectively. These two matrices will eventually be used to determine the spatial coordinates and pose $(x_t^m, y_t^m, z_t^m, \text{ and } \theta_t^m)$ of the UAV in the pseudo map $\boldsymbol{O}_g$.

$$[\boldsymbol{R}_A^c | \boldsymbol{T}_A^c] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} = [\boldsymbol{r}_1 \quad \boldsymbol{r}_2 \quad \boldsymbol{r}_3 \quad \boldsymbol{T}_A^c], \tag{8}$$

where r_{ij} is an element of the rotation matrix $\boldsymbol{R}_a^c$, and (t_x, t_y, t_z) are the elements of translation matrix $\boldsymbol{T}_A^c$. To determine the unknown variables of Equation (8), the camera intrinsic parameter matrix $\boldsymbol{A}$, which can represent optical characteristics of the camera such as focal length, principal point, skewness, etc., should be implemented. $\boldsymbol{A}$ is expressed in Equation (9):

$$\boldsymbol{A} = \begin{bmatrix} \alpha_u & s & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{9}$$

where α_u and α_v are the camera scale factors, (u_0, v_0) is the principal point, and s is the skewness coefficient of the camera. For this purpose, the pinhole camera projection model is used in Equation (10)[37]:

$$\begin{bmatrix} \bar{u}_p \\ \bar{v}_p \\ 1 \\ \bar{w} \end{bmatrix} = \boldsymbol{A}[\boldsymbol{r}_1 \quad \boldsymbol{r}_2 \quad \boldsymbol{r}_3 \quad \boldsymbol{T}_A^c] \begin{bmatrix} \bar{X}_p \\ \bar{Y}_p \\ 0 \\ 1 \end{bmatrix} \tag{10}$$
$$\begin{bmatrix} \bar{u}_p \\ \bar{v}_p \\ \bar{w} \end{bmatrix} = \boldsymbol{A}[\boldsymbol{r}_1 \quad \boldsymbol{r}_2 \quad \boldsymbol{T}_A^c] \begin{bmatrix} \bar{X}_p \\ \bar{Y}_p \\ 1 \end{bmatrix},$$

where $\boldsymbol{r}_1$, $\boldsymbol{r}_2$, and $\boldsymbol{r}_3$ are the columns 1, 2, and 3 of $\boldsymbol{R}_A^c$, and $\bar{w}$ is a scaling factor for image projection for the UAV camera. Let $\boldsymbol{L} = \boldsymbol{A}[\boldsymbol{r}_1 \quad \boldsymbol{r}_2 \quad \boldsymbol{T}_A^c]$ so that Equation (10) is written in the form shown in Equation (11):

$$\begin{bmatrix} \bar{u}_p \\ \bar{v}_p \\ \bar{w} \end{bmatrix} = \boldsymbol{L} \begin{bmatrix} \bar{X}_p \\ \bar{Y}_p \\ 1 \end{bmatrix}, \text{ where } \boldsymbol{L} = \boldsymbol{A}[\boldsymbol{r}_1 \quad \boldsymbol{r}_2 \quad \boldsymbol{T}_A^c]$$
$$\begin{bmatrix} \bar{u}_p \\ \bar{v}_p \\ \bar{w} \end{bmatrix} = \begin{bmatrix} L_{11} & L_{12} & L_{13} \\ L_{21} & L_{22} & L_{23} \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} \bar{X}_p \\ \bar{Y}_p \\ 1 \end{bmatrix}, \tag{11}$$

where $\boldsymbol{L}$ is a homography matrix for camera projection, and L_{ij} are the elements of $\boldsymbol{L}$. By solving for u_p and v_p, a system of linear equations can be obtained for each point, as shown in Equation (12):

$$u_p = \frac{\bar{u}_p}{\bar{w}} = \frac{L_{11}\bar{X}_{ap} + L_{12}\bar{Y}_{ap} + L_{13}}{L_{31}\bar{X}_{ap} + L_{32}\bar{Y}_{ap} + L_{33}},$$
$$v_p = \frac{\bar{v}_p}{\bar{w}} = \frac{L_{21}\bar{X}_{ap} + L_{22}\bar{Y}_{ap} + L_{23}}{L_{31}\bar{X}_{ap} + L_{32}\bar{Y}_{ap} + L_{33}}, \tag{12}$$

which can be rearranged to Equation (13):

$$u_p\bar{X}_{ap}L_{31} + u_p\bar{Y}_{ap}L_{32} + u_pL_{33} - L_{11}\bar{X}_{ap} - L_{12}\bar{Y}_{ap} - L_{13} = 0,$$
$$v_p\bar{X}_{ap}L_{31} + v_pL_{32}\bar{Y}_{ap} + v_pL_{33} - L_{21}\bar{X}_{ap} - L_{22}\bar{Y}_{ap} - L_{23} = 0, \tag{13}$$

which can be written in matrix form as shown in Equation (14):

$$\begin{bmatrix} \bar{M}_p & 0^T & -u_p\bar{M}_p \\ 0^T & \bar{M}_p & -v_p\bar{M}_p \end{bmatrix} \cdot \vec{L} = 0. \quad (14)$$

Here, $\vec{L} = [L_{11}, L_{12}, L_{13}, L_{21}, L_{22}, L_{23}, L_{31}, L_{32}, L_{33}]^T$. For every corner point M_p, two equations are obtained; thus, four points are used to compute the values for $\vec{L}$ using single value decomposition (SVD). Next, the values for $\vec{L}$ provided by SVD are taken as an initial solution and further refined by using iterative nonlinear minimization using the Levenberg-Marquardt algorithm.[44] This algorithm is nonlinear and provides robust pose estimation in the presence of Gaussian noise by minimizing the reprojection error, as shown in Equation (15):

$$\vec{L} := \min_{\text{error}} \left(\sum_{\text{iterations}} \left\| m_p - \hat{m}_p \right\|^2 \right) \quad (15)$$

where $\hat{m}_p$ is the predicted reprojection point obtained using the estimated $\vec{L}$. Finally, the elements of the required extrinsic parameters $\left[R_A^c \middle| T_A^c\right]$ of the camera are computed using Equations (16) and (17):

$$[r_1 \quad r_2 \quad T_A^c] = A^{-1} \cdot L \quad (16)$$

$$r_3 = r_1 \times r_2 \quad (17)$$

R_A^c is first transformed to obtain the orientation of camera R_c^A in O_A using the inverse operation. Since rotation matrices are orthogonal, the transpose of the rotation matrix is the inverse of that matrix, as shown in Equation (18). Similarly, the position vector of the camera with respect to ArUco is obtained using Equation (19).

$$R_c^A = (R_A^c)^T, \quad (18)$$

$$T_c^A = -T_A^c, \quad (19)$$

where T_c^A is the position vector of camera in O_A. Now we have the orientation of camera R_c^A with respect to O_A, and a series of rotation transformations is required to get the pose of the UAV in O_g, as shown in Figure 12.

The series of transformations can be shown in Equation (20):

$$R_d^g = R_c^d \cdot R_A^g \cdot R_c^A, \quad (20)$$

where R_A^g and R_c^d are 3×3 rotation matrices that can be directly computed using the general form of rotation matrices and known angle of rotation.[45] Here, R_A^g aligns O_A with O_g, and R_c^d aligns O_c with using the known orientation of ArUco and camera (see Figures 11 and 12). Equation (21) will yield the required global position of the UAV in the pseudo map:

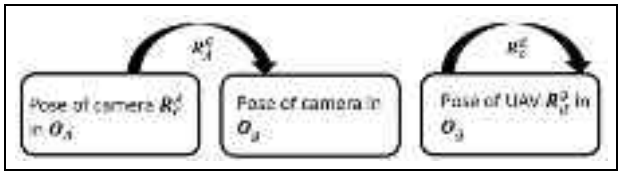


Figure 12. Rotation transformations to find pose of UAV.
UAV: unmanned aerial vehicle.

$$V_g^d = \begin{bmatrix} x_t^m \\ y_t^m \\ z_t^m \end{bmatrix} = R_d^g \cdot T_c^A + V_g^A \quad (21)$$

where $V_g^A = [x_{gA}, y_{gA}, z_{gA}]^T$ is the global position vector of the ArUco marker stored in the marker library in the ground station. Then, θ_t^m can be calculated directly from R_d^g as shown in Equation (22):

$$\theta_t^m = atan2\left(R_{d_{11}}^g, R_{d_{21}}^g\right). \quad (22)$$

Here, for *atan*, the *atan*2 function included in the Math Library in Python[46] is used to calculate the yaw angle in range $[-\pi, \pi]$ radians; $R_{d_{11}}^g$ is the first item in the first row, and $R_{d_{21}}^g$ is the first element of the second row of R_d^g.

Crack damage segmentation using STRNet

For a damage detection task, we recently used an existing algorithm called STRNet,[18] which is an advanced crack segmentation algorithm. STRNet is a deep learning network for crack segmentation in a complex background for real-time processing (i.e., 49 FPS for a relatively large size input image (1024×512)) using a self-attention-based decoder, as shown in Figure 13. The input image I_t^{m2} (1024×512) from Camera_2, as shown in Figure 1 goes through the first convolution block, which involves convolution and batch normalization[47] with Hswish[48] activation function. Next, the extracted feature map by the convolution block is inputted into the semantic transformer representation (STR) module, which is repeated 11 times. After that, max pooling is performed, and the result is inputted into the attention decoder that we developed. The attention decoder result after two repetitions are upsampled as the original size of the input image. All the green lines express skipped connections to keep multi-dimensional features throughout the process of STRNet. In addition, the outputs of the upsampling layer, first convolution block, and coarse upsampling block are concatenated and go through pointwise convolution (PW) to produce an output. The output provides segmented cracks at the pixel level. In the

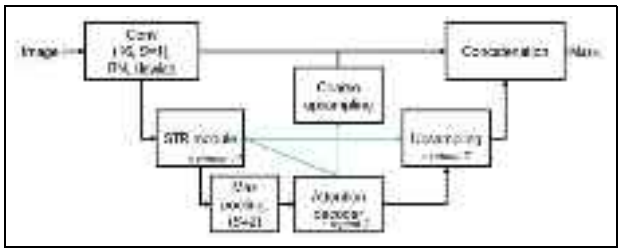


Figure 13. STRNet architecture.
STRNet: semantic transformer representation network.

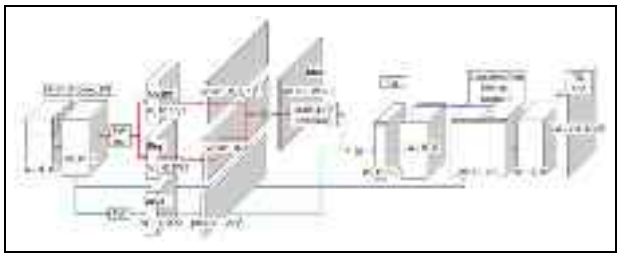


Figure 14. Attention decoder for STRNet.
STRNet: semantic transformer representation network.

following subsections are brief descriptions of the STR module and attention decoder.

STR module. CNNs use many convolution layers to extract an object-focused feature map. The computational cost of a convolutional operator is usually dependent on kernel size (k), input channel (C), input width (W), input height (H), and output channel (O). The computational costs of a traditional (T) convolution, depthwise (DW) convolution, PW, and their combination are calculated as

$$T = k^2 CWHO \tag{23}$$

$$\mathrm{DW} = k^2 CWH \tag{24}$$

$$\mathrm{PW} = CWHO \tag{25}$$

$$\mathrm{DW} + \mathrm{PW} = CHW\left(k^2 + O\right) \tag{26}$$

As shown in Equations (23) to (26), the cost is heavily affected by C, H, and W. To reduce computational costs, a pooling layer or convolution with two strides is usually applied. When we increase the size of the stride, W and H from Equations (23) and (26) are reduced. Therefore, proper sizes of the strides in each layer and operator are important to reduce the computational cost of a specific network, but the increase of stride may cause the loss of important features. This is a very critical problem in crack segmentation, since crack lines are thin, and specific feature loss may critically reduce the performance of the segmentation. To overcome the issue, an STR module was developed. The STR module consists of "Compress," "Attention," and "Restoration" modules. The base module is mainly composed of the PW and DW convolution blocks. As shown in Figure 13 Compress module applies DWs to increase stride sizes, and the compressed feature map is passed to the Attention module which includes a squeeze and excitation block (SE block).[49]

In the SE block, to produce the attention map, global average pooling is applied for the input feature map, and two linear functions and all activation functions, such as rectified linear unit[50] and H-sigmoid,[51] are performed as a squeezing process. Then, the squeezed feature map is expanded by coping of the squeezed map to the input size of the SE block to apply the matrix multiplication (■). Finally, it is multiplied by the input of the SE block. The compress and restoration block (CRB) compresses the width and height of the input feature map and recovers the compressed size of the feature map to the original input feature map of the STR module by combining the reserved features using the skipped connection. The concatenation operation helps prevent a vanishing gradient and the loss of important features.

Attention decoder. After the STR encoder, we apply the attention decoder, as shown in Figure 14. The original concept of attention comes from a previous study.[52] Recently, an improved attention-based model, transformer, proposed by Vaswani et al.,[53] applies multi-head attention using Key, Query, and Value. Query, Key, and Value are calculated using scaled dot-product to find the attention map and enhance the important features. This can be expressed by Equation (27)

$$\mathrm{Attention} = \mathrm{Softmax}\left(\frac{\mathrm{Query}^T * \mathrm{Key}}{\sqrt{D}}\right) * \mathrm{Value}^T \tag{27}$$

Data generation for training of STRNet. Deep learning-based semantic segmentation methods have been actively adopted in pixel-level crack segmentation research in the last few years. However, annotating highly accurate segmentation data entails high labor costs because of labeling ground truth. For example, the annotation at the pixel level for PASCAL VOC is evaluated to be 239.7 seconds per image.[54] Annotating crack images take approximately 20–40 min.[55] To reduce the time required to prepare the dataset, we considered two options: (1) a public dataset and (2) raw images only.

Public concrete crack datasets exist.[10,56–58] Although Choi and Cha[10] created a dataset using large-sized images, most other public crack datasets contain small-sized images, and the image and mask occasionally

Table 1. Crack datasets for training and testing (Kang and Cha, 2021).

Dataset	Train	Test	Total
Crack 1748 (1024 × 512)	1203	545	1748
UAV 54 (1024 × 512)	—	54	54
Total	1203 (12,030)	599	1802

UAV: unmanned aerial vehicle.

have errors. Furthermore, almost all public crack datasets have images with very simple backgrounds. For this reason, many public crack images were not selected for our training dataset, and a new dataset was developed[18] (see Table 1).

STRNet training and evaluation. In this experiment, STRNet was trained using the data tabulated in Table 1. The Python programming language[46] with the Pytorch 1.6 deep learning library[59] was applied to implement STRNet. It was trained with four Titan XP graphics processing units (GPUs). The workstation had an Intel Core i7-6850 K central processing unit (CPU) and 128 GB memory. To train our models, the Nvidia Apex distributed data parallel training library was used to utilize the multi-GPUs. The input image size was 1024 × 512, which is randomly cropped if the image size is greater than the input size. The use of proper loss function is critical; therefore, we investigated several loss functions, such as cross-entropy loss, dice cross-entropy loss, and mIoU. Eventually, the focal Tversky loss function showed the best performance.[60] The formulation of the loss function is presented in Equation (28)

$$\text{Focal} - \text{Tversky loss} = \left(1 - \frac{\text{TP} + \varepsilon}{\text{TP} + \theta * \text{FP} + \mu * \text{FN} + \varepsilon}\right)^{\tau} \tag{28}$$

The hyperparameters ε, θ, μ, and τ are defined as 0.5, 0.5, 1.0, and 1.3, respectively. θ scales FP (false positives), while μ scales FN (false negatives). ε is a small positive number for smoothing of the training process. τ increases focal loss. These values were investigated by trial and error.

An Adam optimizer was applied as the optimizer, and the hyperparameters of β_1 and β_2 were set as 0.9 and 0.999, respectively.[61] To validate the superior performance of STRNet, algorithms developed in previous research were also tested for comparison, such as Attention U-Net, MobileNetV3 Large,[51] and Deeplab v3+.[62] We used all the same training data and testing data for a fair comparison. The evaluation results of the four networks are represented in Figure 15.

To calculate mIoU, we used Equation (29) since it can properly consider FP and FN compared to the other recall, precision, and F1 scores

$$\text{mIoU} = \text{mean}\left(\frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}}\right) \tag{29}$$

Using the dataset presented in Table 1, we conducted extensive training and testing, and the results are presented in Table 2. STRNet achieved the best mIoU (92.5%) and F1 score (92.08%) from 545 various testing images having complex scenes and the best mIoU (91.8%) and F1 score (90.7%) from 54 images from the UAV camera. Some of the examples are illustrated in Figure 15.

MobileNetV3 included a false positive object in images a, c, and d. Deeplab v3+ showed more severe false positive detection in all the images. Attention U-Net detected expansion joints as cracks in images a, b, and c. The difficulty of the crack 1748 dataset lies in complex scenes and various shapes of crack-like objects. STRNet successfully overcame most of these challenging problems and showed very accurate segmentation results. Based on these achievements, STRNet was implemented on the images taken from the UAV camera for inspection. The results are presented in Section "Case studies."

Table 2. Performance comparison of different models for crack detection on UAV images and crack segmentation on ground images.

Model	Crack 1748 (545)		UAV 54		Speed
	F1 Score	mIoU	F1 Score	mIoU	V100 (FPS)
MobileNetV3	85.61	**87.1**	74.4	**80.0**	71
Attention U-Net	88.3	**89.1**	80.6	**84**	17
Deeplab v3+	80.36	**83.24**	69.2	**76.9**	30.2
STRNet	92.08	**92.5**	90.7	**91.8**	47.8

FPS: frames per second; mIoU: mean intersection over union; UAV: unmanned aerial vehicle.
The bold values are highlighting mIoU performance of networks.

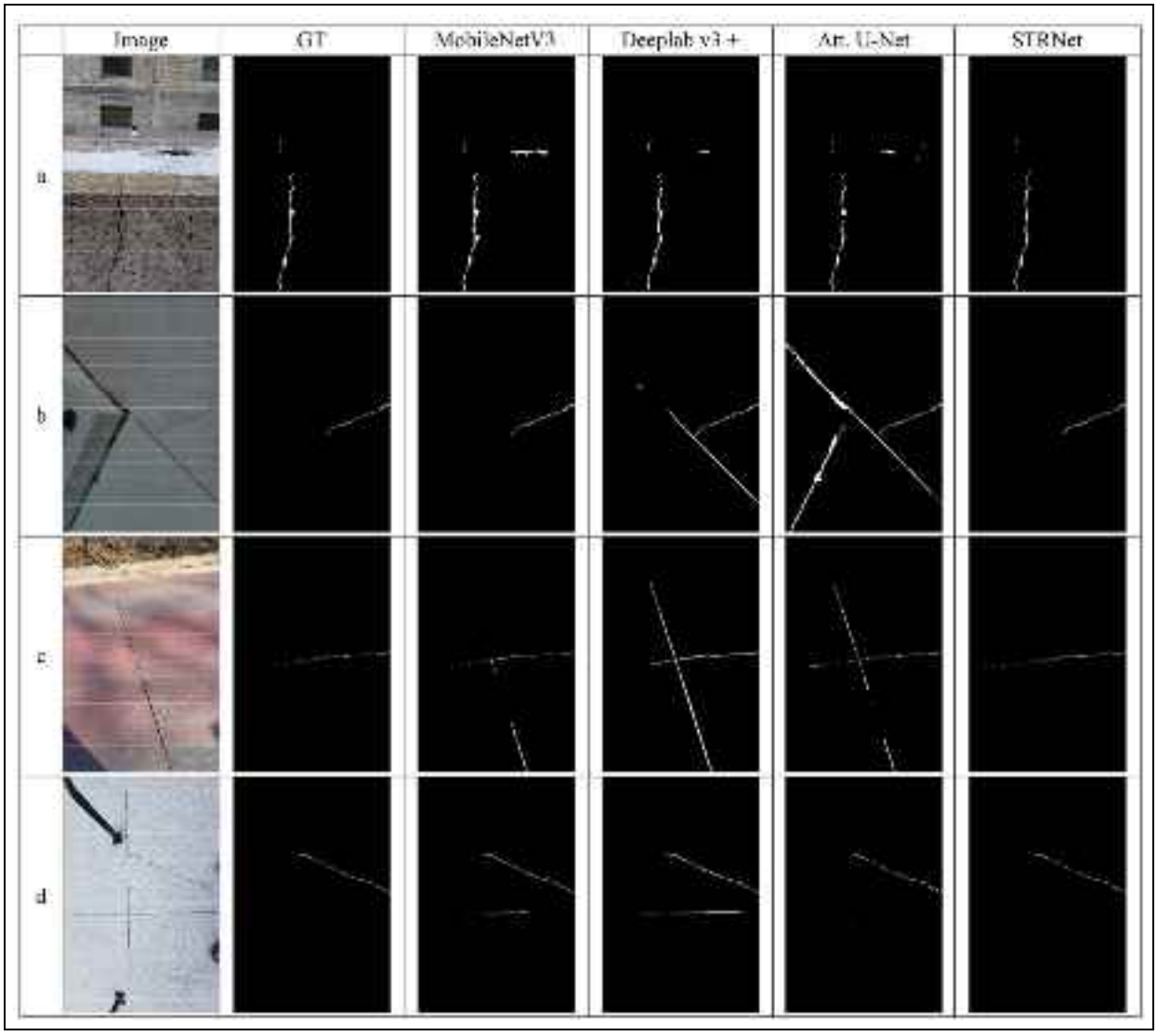


Figure 15. Example results of the comparative studies on complicated images: (a) concrete crack with a building in background, (b) concrete crack on pavement, (c) thin crack on concrete with grass in background and (d) crack with slab joints.

Case studies

Parrot Bebop 2 Power Drone[63] and Anafi[64] (https://www.parrot.com/en/about-parrot) were selected to test autonomous UAVs, as shown in Figure 16. This allows the developer to develop their own controller for autonomous navigation. A laptop computer with an Intel Core i7 2.6 GHz CPU and Nvidia Geforce (https://www.nvidia.com/en-us/) 1060 GPU communicated with the UAV through Wi-Fi-based communication. The front camera of the UAV was used for obstacle detection, and due to the limitation of the payload, a lightweight camera was chosen as the second camera for damage (crack) segmentation. As the second camera, we used the Hawkeye Firefly Micro Cam 160 model, as indicated in the red box in Figure 16. We modified the hardware of the camera to install it on the UAV. Two UAV localization methods, Marvelmind USB for indoor experiments and ArUco marker-based localization for outdoor experiments, were used, as explained in Figure 1 in Section "Methodology." The details of experimental testing and validation for indoor and outdoor environments are explained in the following subsections "Indoor experiments" and "Outdoor experiments," respectively.

Indoor experiments

The proposed OAM was initially tested in an indoor environment with a USB-based localization method. In order to investigate the performance, it was

Figure 16. Hardware settings: (a) USB-based approach with Marvelmind beacon; inspection camera: Firefly Micro Cam 160 model, Bebop Power 2, and; (b) ArUco marker-based approach using an Anafi UAV.
UAV: unmanned aerial vehicle; USB: ultrasonic beacon system.

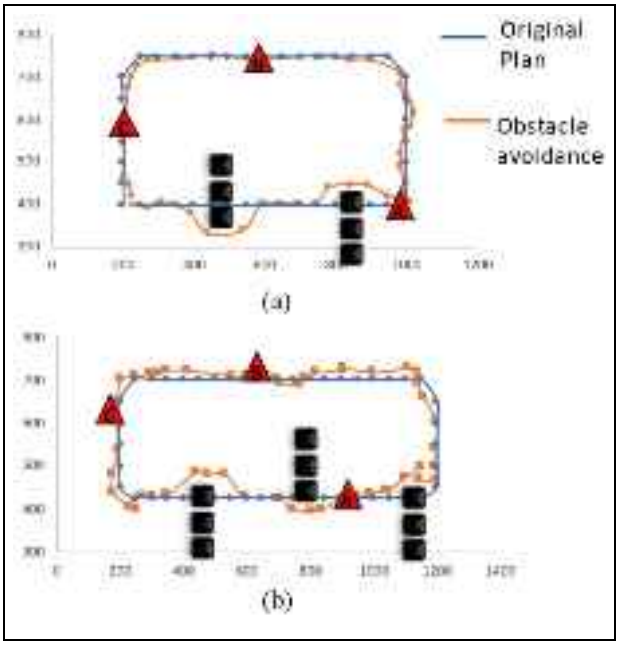


Figure 17. Flight path: (a) obstacle avoidance with two clusters of obstacles, and (b) obstacle avoidance with three clusters of obstacles.

compared with existing OAMs such as the MonodepthV2 (Godard et al., 2019)[32] and AdaBins (Bhat et al., 2020)[33] approaches. Details of the OAM and STRNet experimental tests are provided below.

Obstacle avoidance

In order to conduct experimental tests to validate the proposed obstacle avoidance algorithm and crack damage detection, we generated a planned trajectory for autonomous navigation of the UAV in the laptop

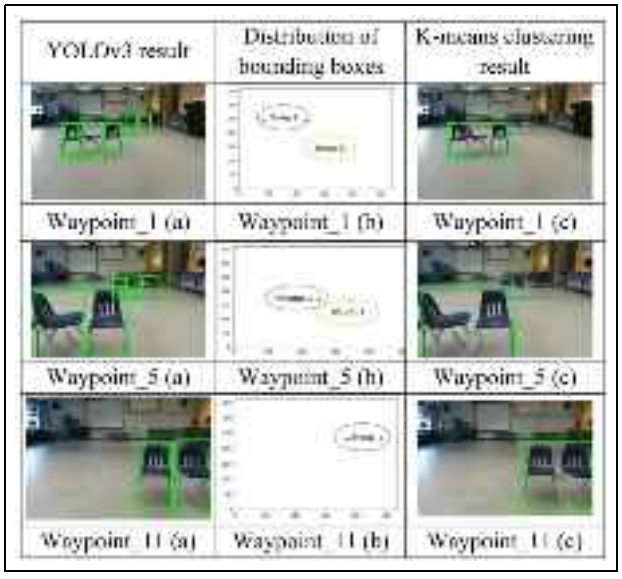


Figure 18. OAM performances based on YOLOv3 and K-means clustering at waypoints 1, 5, and 11.
OAM: obstacle avoidance method.

ground station by assigning a series of waypoints (expressed as blue lines, as shown in Figure 17). The distance between each waypoint was set as 50 cm. Two and three groups of obstacles were set, which are illustrated as black rectangular boxes. The locations of detected cracks in the concrete floor are illustrated as red triangles. Specific waypoints were expressed as red circles in obstacle scenario 1. As shown in Figure 17, two cases of obstacle scenarios were well avoided by using our proposed OAM. The orange lines express the trajectory to avoid obstacles. Also, during the navigations, all three cracks in the concrete floor were detected well. All the dimensions are expressed as cm units.

There were three chairs in each group of obstacles, and the proposed OAM (see Section "Obstacle clustering") operated well during the autonomous navigation. As shown in Figure 18, YOLOv3 detected obstacles successfully, and the K-means clustering algorithm created a group of boxes based on features and converted them into a single obstacle box at waypoints 1, 5, and 11.

As comparative study, our approach was compared to existing deep learning-based monocular depth estimation algorithms such as MonodepthV2 (Godard et al., 2019)[32] and AdaBins (Bhat et al., 2020).[33] Using the two tests, 20 image frames with obstacles in the scenes from each autonomous navigation were selected to compare the performance of the obstacle identification/clustering. A total of 40 image frames were used as the input for the three methods and tested, as shown in Figure 19.

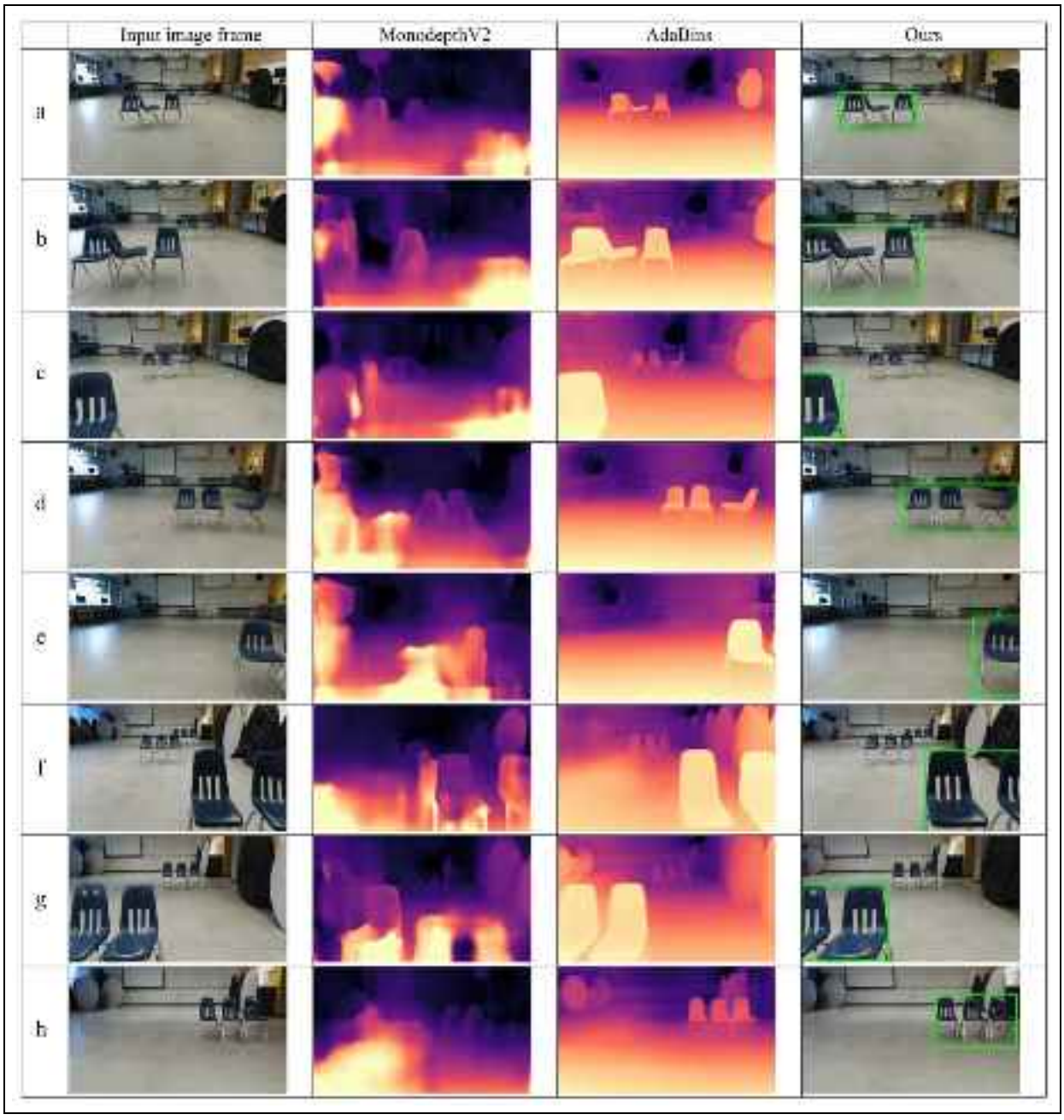


Figure 19. Detected obstacles in three methods: (a) Farther away obstacles in center, (b) Closer obstacles, (c) Multiple obstacle groups, (d) Farther away obstacles to right, (e) Partially visible obstacle, (f) Two closer obstacles, one partially visible, (g) Two closer obstacles, one partially visible to left, and (h) Three obstacles fully visible to right grouped.

Our object clustering algorithm and AdaBins worked well to identify obstacles in various scenarios in terms of location within the image frame. However, MonodepthV2 failed to identify the chairs as shown in Figure 19(a) to (d). Furthermore, empty spaces were detected as obstacles consistently in Figure 19(a) to (h).

While MonodepthV2 failed to provide the proper quality of the depth map to identify obstacles, AdaBins and Ours achieved very successful results. To evaluate

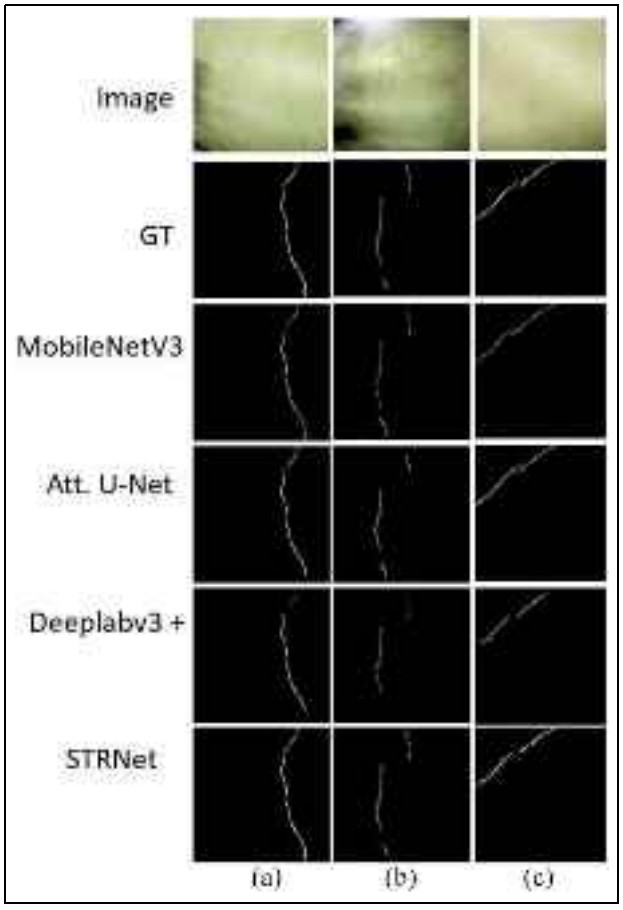


Figure 20. Comparison between MobileNetV3, Attention U-Net, Deeplab v3 + , and STRNet: (a) crack image 1, (b) crack image 2, and (c) crack image 3.
STRNet: semantic transformer representation network.

the performances, we use the three most common evaluation metrics: precision, recall, and $F1$ score, as expressed in Equations (30)–(32). True positive, true negative, false positive, and false negative cases are abbreviated to TP, TN, FP, and FN, respectively.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{30}$$

$$\text{Recall} = \left(\frac{\text{TP}}{\text{TP} + \text{FN}}\right) \tag{31}$$

$$F1\ \text{score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{32}$$

All the calculated evaluation metrics are tabulated in Table 3. The results of 40 images were manually checked. There were two obstacles in scenario 1 and three obstacles in scenario 2, as shown in Figure 17, which appeared in the 40 image frames 95 times. AdaBins and Ours showed satisfactory results in these two cases of autonomous navigation with different obstacle scenarios, as presented in Table 3.

Table 3. Evaluation result of obstacle state for each algorithm

Algorithm	Precision	Recall	F1_score	FPS
AdaBins	1.00	1.00	1.00	0.3
MonodepthV2	0.76	0.78	0.77	0.5
Ours	1.00	1.00	1.00	10

FPS: frames per second.

Figure 21. Outdoor experimental setup using the ArUco-based navigation system with chairs as obstacles.

However, the processing speed of the AadBins was just 0.3 FPS, which may delay the overall inspection process due to the low speed of the OAM, since videos are commonly 30 FPS. However, our algorithm was still 10 FPS using an old 1060 GPU. Also, as shown in Figure 19(a), (b), and (d), AdaBins could not detect the legs of the chairs. It can be potentially dangerous in infrastructure inspections if small, thin obstacles are not properly detected during navigation for monitoring.

STRNet results for indoor case. Figure 20 shows the three different cracks presented in Figure 17 as red triangles (a, b, c). The cracks in the images from the UAV camera are seen as very unclear and blurry due to the vibration of the UAV. The results of the Deeplab v3 + and MobileNet v3 suffered from loss of the crack pixels throughout the line of the cracks (Figure 20(a) and (c)). Attention U-Net showed good results in image c, but it missed the end of the crack in images a and b. Among these algorithms, STRNet showed very good performance in all images, with 91.8% mIoU.

Outdoor experiments

After obtaining successful results from indoor experiments, outdoor experiments were performed in a multi-

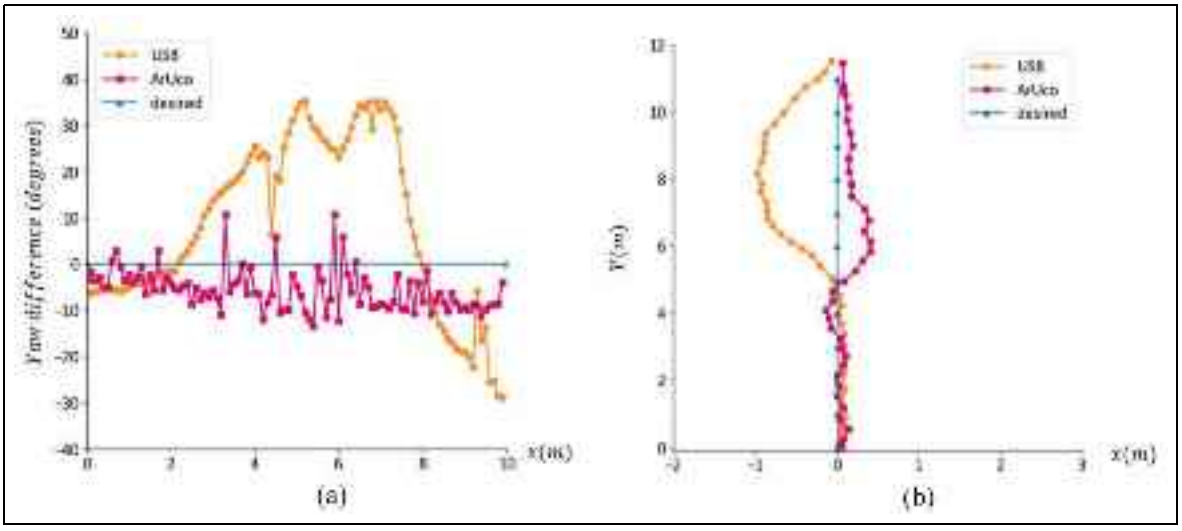


Figure 22. Comparing the performance of (a) deviation in yaw control and (b) deviation in path control for the ArUco marker and USB-based localization methods in an outdoor parking structure.
USB: ultrasonic beacon system.

level parking structure in the Fort Garry campus of the University of Manitoba. The top floor was used to conduct the autonomous UAV experiments. The wind direction was along the southeast, with wind gusts of more than 30 km/h.

The average height of the UAV's planned waypoint was 7 m from ground level. This natural, real-world environment was used to validate the robustness of the developed systems. ArUco markers were installed at every 1 m on the ceiling in the designated section, as shown in Figure 21. For safety, nets were installed instead of windows. In the experimental setup, the UAV was under the impact of strong to medium wind gusts. Different sets of planned waypoints were used to confirm the validity of the proposed method of UAV control via ArUco markers. The overall outdoor experiment area is shown in Figure 21. During the experiments in the parkade garage, the lighting conditions were challenging due to the low light levels and the presence of occluding lights. These conditions can affect the performance of a camera-based system, as the low light levels can make it difficult for the camera to capture clear images, and the occluding lights can create glare and shadows that may obscure objects of interest. These markers are designed to be easily detected by a camera system, even when the lighting conditions are poor.

Yaw control of UAV. The most important feature of using computer vision-based ArUco marker-based localization is the robust estimation of yaw θ_t^m, which is critical to follow a set of *planned* waypoints $(x_t^p,\ y_t^p,\ z_t^p,$ and $\theta_t^p)$. In our outdoor experiments, tests were conducted to show the robustness of yaw control using ArUco markers under a GPS-denied and high magnetic interference environment in a parking structure. The UAV was given the task of moving in a straight line for 10 meters while maintaining an angle of 0° along the planned waypoints shown in Figure 22(a).

It was observed that the USB sensors resulted in poor readings when close to steel girders due to magnetic interference. The noisy and unreliable readings from the USB resulted in poor control of the UAV. As a result, the UAV deviated from the planned waypoints when relying on USB data, as shown in Figure 22(b). The yaw deviation $\left(\theta_t^m - \theta_t^p\right)$ for USB was in the range of $[-25°$ to $31°]$ with an SD of 16.43° and a root mean square deviation (RMSD) of 18.14°. Similarly, the RMSD value for the flight path while using USB was 0.48 m.

On the other hand, the ArUco marker-based localization was not impacted by magnetic interference and provided the robust yaw values required to control the UAV under the same challenging and complex environment. The deviation $\left(\theta_t^m - \theta_t^p\right)$ in using ArUco markers was limited to within $\left[-13^\circ,\ 10^\circ\right]$ with an SD of 4.974° and an RMSD value of 7.89°. The RMSD value for path following with ArUco markers was 0.114 m. Thus, in the outdoor environment, the ArUco marker-based localization decreased the error in yaw control by 60.45% and decreased the error in path following by 67.29%.

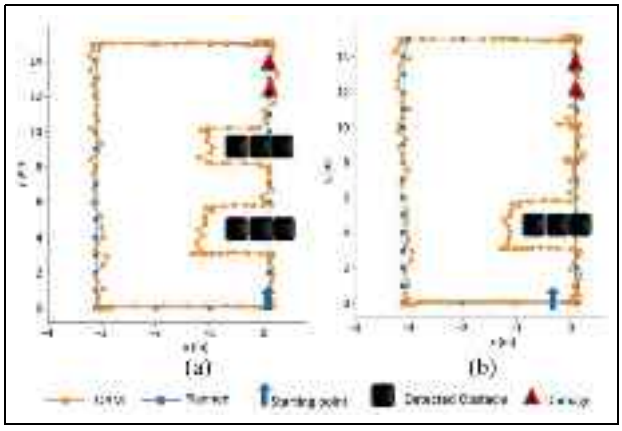


Figure 23. OAM path versus planned path when navigating through: (a) one group of obstacles, and (b) two groups of obstacles.
OAM: obstacle avoidance method.

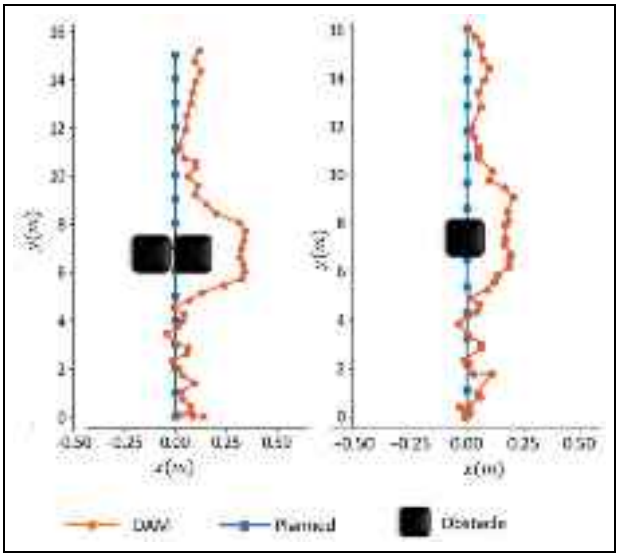


Figure 24. Proposed OAM performance for one and two chairs.
OAM: obstacle avoidance method.

Obstacle avoidance. Based on the comparative studies of the performances of the USB-based and ArUco marker-based localization methods, we chose ArUco marker-based localization for outdoor experimental validation of our proposed OAM.

To avoid obstacles such as chairs, the UAV camera must be able to capture images or video frames from the front side. For this, the camera feed I_t^{m1} is used to detect obstacles using YOLOv3, as discussed in Section "Obstacle avoidance method." Two sets of obstacle configurations were used to verify the performance of

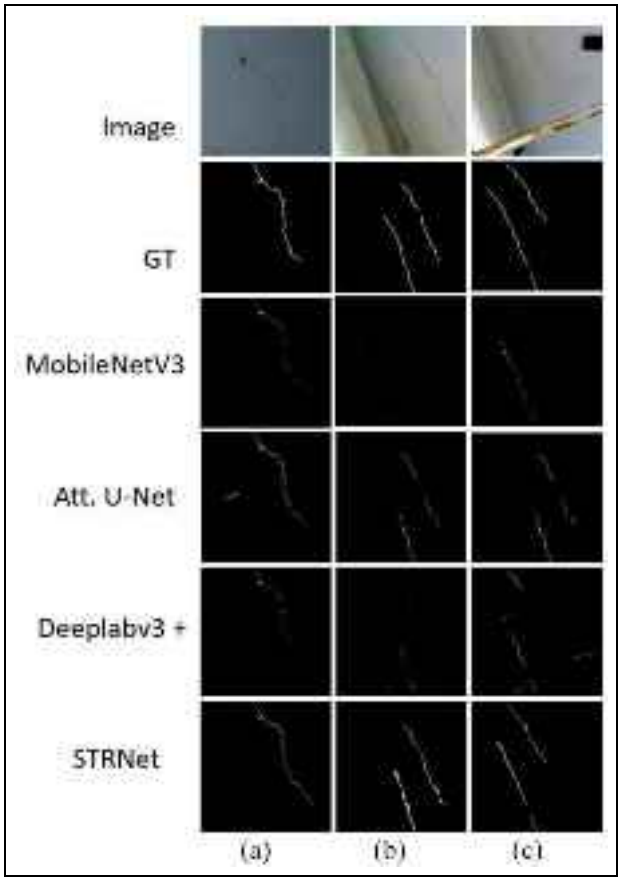


Figure 25. Comparison of crack detection methods using MobileNetV3, Attention UNet, Deeplab v3+, and STRNet on UAV captured images of parkade ceiling: (a) Crack 1, (b) Crack 2, and (c) Crack 3 are highlighted for evaluation during autonomous inspection.

the OAM algorithm. In this experiment, the planned route was a 3.1 m×15 m rectangular path for the collection of data from the ceiling of parkade structure. Sets of obstacles were placed at 4 m and at 9 m along the y-axis, as shown in Figure 23. The resulting path followed by the UAV is shown in Figure 23 as an orange-colored line. Both obstacles were successfully avoided by the autonomous UAV. The red triangle markers of A and B in Figure 23 are the locations of the detected concrete slap cracks analyzed by STRNet, which is discussed below. Moreover, as shown in Figure 24, the OAM was able to successfully detect and avoid one and two chairs placed in the planned way point of the UAV using the real-time object detection algorithm. These results show that the UAV can successfully navigate and avoid obstacles, regardless of the number of chairs present, and demonstrates the robustness and flexibility of our UAV's obstacle avoidance capabilities.

STRNet results for outdoor case. In Figure 25, the performance of STRNet on the data collected from the outdoor parkade is shown. Three images of two separate crack images are presented in Figure 25; their positions are shown as red triangles in Figure 23. Similar to the indoor experiments, the Deeplab v3 and MobileNetV3 performances were degraded as compared to that of Attention U-Net (Figure 25(b) and (c)). Overall, STRNet performed consistently well over all testing datasets and outperformed all the models in comparison. The crack images of Figure 25(a) are from the area indicated by red triangle A in Figure 23, and those in Figure 25(b) and (c) from that of red triangle B in Figure 23.

Conclusion and future direction

To develop a reliable autonomous UAV system for SHM, we developed a framework for obstacle-avoiding autonomous flight methods through fiducial marker-based UAV localization. The contributions of the paper are as follows:

(1) A complex autonomous UAV system for SHM was developed by careful integration of a new OAM, a new localization method, and a state-of-the-art real-time crack segmentation method.
(2) A new OAM was developed with YOLOv3 (as an example network, but it can be replaced by any recent version of YOLO) and an obstacle clustering method based on the K-mean clustering method.
(3) The developed OAM was compared to existing methods, and it outperformed with a higher FPS and robust obstacle detection compared to those of the traditional OAM methods.
(4) A new autonomous UAV localization method was introduced for autonomous UAVs for SHM with fiducial ArUco markers. Due to the nature of the ArUco marker, it does not require regular battery charging and is not vulnerable to environmental electromagnetic fields.
(5) The ArUco marker-based method decreased localization error by 67.29% and yaw control error by 60.45% compared to those of the USB method, respectively.
(6) The new OAM, fiducial ArUco marker-based autonomous UAV system, was implemented in indoor and outdoor environments as GPS-denied areas.
(7) The developed autonomous UAV system including a new OAM and localization method was integrated with a state-of-the-art crack segmentation network (STRNet).
(8) STRNet[18] showed superior performance (mIoU 92.5%) compared to recently developed deep CNNs (mIoU 89.1%) for parking structure crack segmentation.

In future work, we plan to integrate new hybrid techniques, such as 3D sensor fusion, to estimate the distance of unknown obstacles for autonomous UAVs and integrate the developed method with a digital twin for real-time damage mapping and management.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: The research presented in this paper was supported by an NSERC Discovery grant (RPGIN-2016-05923), a Research Manitoba Innovation Proof-of-Concept Grant (4914), and a CFI JELF grant (37394).

ORCID iD

Young Jin Cha https://orcid.org/0000-0002-0738-5615

References

1. Cha YJ, Choi W and Büyüköztürk O. Deep learning-based crack damage detection using convolutional neural networks. *Comput-Aided Civ Infrastruct Eng* 2017; 32(5): 361–378.
2. Cha YJ, Choi W, Suh G, et al. Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types. *Comput-Aided Civ Infrastruct Eng* 2018; 33(9): 731–747.
3. Lin YZ, Nie ZH and Ma HW. Structural damage detection with automatic feature-extraction through deep learning. *Comput-Aided Civ Infrastruct Eng* 2017; 32(12): 1025–1046.
4. Lee K, Lee S and Kim HY. Bounding-box object augmentation with random transformations for automated defect detection in residential building façades. *Autom Constr* 2022; 135: 104138.
5. Huynh TC, Park JH, Jung HJ, et al. Quasi-autonomous bolt-loosening detection method using vision-based deep learning and image processing. *Autom Constr* 2019; 105: 102844.
6. Ronneberger O, Fischer P, and Brox T. U-net: Convolutional networks for biomedical image segmentation. In: *Medical Image Computing and Computer-Assisted InterventionMICCAI 2015: 18th International Conference*, Munich, Germany, 5–9 October 2015, pp.234–241. Springer International Publishing.

7. Cheng J, Xiong W, Chen W, et al. Pixel-level crack detection using U-net. In: *TENCON 2018-2018 IEEE region 10 conference*, Jeju, Korea (South), 2018, pp. 0462–0466. New York: IEEE.
8. Escalona U, Arce F, Zamora E, et al. Fully convolutional networks for automatic pavement crack segmentation. *Comput y Sist* 2019; 23(2): 451–460.
9. Dais D, Bal İE, Smyrou E, et al. Automatic crack classification and segmentation on masonry surfaces using convolutional neural networks and transfer learning. *Autom Constr* 2021; 125: 103606.
10. Choi W and Cha YJ. SDDNet: Real-time crack segmentation. *IEEE Trans Ind Electron* 2019; 67(9): 8016–8025.
11. Saleem MR, Park JW, Lee JH, et al. Instant bridge visual inspection using an unmanned aerial vehicle by image capturing and geo-tagging system and deep convolutional neural network. *Struct Health Monit* 2021; 20(4): 1760–1777.
12. He K, Gkioxari G, Dollár P, et al. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, Venice, Italy, 22–29 October 2017, pp. 2961–2969. https://iccv2017.thecvf.com/.
13. Yu Y, Rashidi M, Samali B, et al. Crack detection of concrete structures using deep convolutional neural networks optimized by enhanced chicken swarm algorithm. *Struct Health Monit* 2022; 21: 2244–2263.
14. Abdeljaber O, Avci O, Kiranyaz S, et al. Real-time vibration-based structural damage detection using one-dimensional convolutional neural networks. *J Sound Vib* 2017; 388: 154–170.
15. Ai D, Mo F, Han Y, et al. Automated identification of compressive stress and damage in concrete specimen using convolutional neural network learned electromechanical admittance. *Eng Struct* 2022; 259: 114176.
16. Ai D and Cheng J. A deep learning approach for electromechanical impedance based concrete structural damage quantification using two-dimensional convolutional neural network. *Mech Syst Signal Process* 2023; 183: 109634.
17. Potenza F, Rinaldi C, Ottaviano E, et al. A robotics and computer-aided procedure for defect evaluation in bridge inspection. *J Civ Struct Health Monit* 2020; 10(3): 471–484.
18. Kang DH and Cha YJ. Efficient attention-based deep encoder and decoder for automatic crack segmentation. *Struct Health Monit* 2021; 21: 2190–2205.
19. Bao Y and Li H. Machine learning paradigm for structural health monitoring. *Struct Health Monit* 2021; 20(4): 1353–1372.
20. Xu Y, Bao Y, Chen J, et al. Surface fatigue crack identification in steel box girder of bridges by a deep fusion convolutional neural network based on consumer-grade camera images. *Struct Health Monit* 2019; 18(3): 653–674.
21. Kim H, Lee J, Ahn E, et al. Concrete crack identification using a UAV incorporating hybrid image processing. *Sensors* 2017; 17(9): 2052.
22. McGuire M, Rys MJ and Rys A. *A study of how unmanned aircraft systems can support the Kansas Department of Transportation's efforts to improve efficiency, safety, and cost reduction*. Topeka, Kansas: Kansas Department of Transportation, 2016.
23. Benz C, Debus P, Ha HK, et al. (2019). Crack segmentation on UAS-based imagery using transfer learning. In: *2019 International Conference on Image and Vision Computing New Zealand (IVCNZ)*, Dunedin, New Zealand, 2–4 December 2019, pp. 1–6, New York: IEEE.
24. Kang D and Cha YJ. Autonomous UAVs for structural health monitoring using deep learning and an ultrasonic beacon system with geo-tagging. *Comput-Aid Civ Infrastruct Eng* 2018; 33(10): 885–902.
25. Ali R, Kang D, Suh G, et al. Real-time multiple damage mapping using autonomous UAV and deep faster region-based neural networks for GPS-denied structures. *Autom Constr* 2021; 130: 103831.
26. González-deSantos LM, Martínez-Sánchez J, González-Jorge H, et al. Payload for contact inspection tasks with UAV systems. *Sensors* 2019; 19(17): 3752.
27. Song S, Jung S, Kim H, et al. A method for mapping and localization of quadrotors for inspection under bridges using camera and 3 D-LiDAR. In: *Proceedings of the 7th Asia-Pacific Workshop on Structural Health Monitoring*, Hong Kong SAR, PR China, 2019. https://www.ndt.net/search/docs.php3?id = 24157
28. Mori T and Scherer S. First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles. In: *2013 IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, 2013, pp. 1750–1757. New York: IEEE. https://doi.org/10.1109/ICRA.2013.6630807
29. Bay H, Ess A, Tuytelaars T, et al. Speeded-up robust features (SURF). *Comput Vis Image Underst* 2008; 110(3): 346–359.
30. Smolyanskiy N, Kamenev A, Smith J, et al. Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, Canada, 2017, pp. 4241–4247. New York: IEEE.
31. Loquercio A, Maqueda AI, Del-Blanco CR, et al. Dronet: Learning to fly by driving. *IEEE Robot Autom Lett* 2018; 3(2): 1088–1095.
32. Godard C, Mac Aodha O, Firman M, et al. Digging into self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, Seoul, Korea (South), 2019, pp. 3828–3838. https://www.computer.org/conferences/cps
33. Bhat SF, Alhashim I and Wonka P. AdaBins: Depth Estimation using Adaptive Bins. arXiv preprint arXiv:2011.14141, 2020. https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.00400
34. Garrido-Jurado S, Muñoz-Salinas R, Madrid-Cuevas FJ, et al. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern*

Recognit 2014; 47(6): 2280–2292. https://doi.org/10.1016/j.patcog.2014.01.005
35. Redmon J and Farhadi A. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018. https://doi.org/10.48550/arXiv.1804.02767
36. Redmon J. Darknet: Open source neural networks in C. http://pjreddie.com/darknet (2016).
37. Zhang Z. Flexible camera calibration by viewing a plane from unknown orientations. In: *Proceedings of the seventh IEEE international conference on computer vision*Kerkyra, Greece, 1999, vol. 1, pp.666–673. New York: IEEE.
38. Kendal D. Measuring distances using digital cameras. *Aust Senior Math J* 2007; 21(2): 24–28.
39. Romero-Ramirez FJ, Muñoz-Salinas R and Medina-Carnicer R. Speeded up detection of squared fiducial markers. *Image Vis Comput* 2018; 76: 38–47.
40. Suzuki S. Topological structural analysis of digitized binary images by border following. *Comput Vis, Graph, Image Process* 1985; 30(1): 32–46.
41. Douglas DH and Peucker TK. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Can Geogr* 1973; 10(2): 112–122.
42. Otsu N. A threshold selection method from gray-level histograms. *IEEE Trans Syst Man Cybern* 1979; 9(1): 62–66.
43. Burger W. *Zhang's camera calibration algorithm: In-depth tutorial and implementation*. Report No. HGB16-05, 2016, pp. 1–6. https://www.researchgate.net/publication/303233579_Zhang's_Camera_Calibration_Algorithm_In-Depth_Tutorial_and_Implementation
44. Levenberg K. A method for the solution of certain non-linear problems in least squares. *Quart Appl Math* 1944; 2(2): 164–168.
45. Blesgen T. On rotation deformation zones for finite-strain Cosserat plasticity. *Acta Mech* 2015; 226(7): 2421–2434.
46. Python 3.6.12. Python. https://www.python.org/ (2020, 10 June 2021).
47. Ioffe S and Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: *Proceedings of the 32nd International Conference on Machine Learning*, Lille, France, vol. 37, pp. 448–456. https://jmlr.org/
48. Avenash R and Viswanath P. Semantic segmentation of satellite images using a modified CNN with hard-swish activation function. In: *VISIGRAPP (VISAPP)*, Prague, Czech Republic 2019, pp. 413–420. https://www.scitepress.org/HomePage.aspx
49. Hu J, Shen L and Sun G. Squeeze-and-excitation networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, Salt Lake City, UT, 2018, pp. 7132–7141. New York: IEEE. https://doi.org/10.1109/CVPR.2018.00745
50. Nair V and Hinton GE. Rectified linear units improve restricted boltzmann machines. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*, 2010, pp. 807–814.
51. Howard A, Sandler M, Chu G, et al. Searching for mobilenetv3. In: *IEEE International Conference on Computer Vision*, Seoul, Korea (South), 2019, pp. 1314–1324. New York: IEEE.
52. Bahdanau D, Cho K and Bengio Y. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv: 1409.0473, 2014. https://doi.org/10.48550/arXiv.1409.0473
53. Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. arXiv preprint arXiv:1706.03762, 2017. https://doi.org/10.48550/arXiv.1706.03762
54. Bearman A, Russakovsky O, Ferrari V, et al. What's the point: Semantic segmentation with point supervision. In: Bastian L, Jiri M, and Nicu S, et al. (eds.) *European conference on computer vision*. Cham: Springer, 2016, pp. 549–565.
55. Kang D, Benipal SS, Gopal DL, et al. Hybrid pixel-level concrete crack segmentation and quantification across complex backgrounds using deep learning. *Autom Constr* 2020; 118: 103291.
56. Shi Y, Cui L, Qi Z, et al. Automatic road crack detection using random structured forests. *IEEE Trans Intell Transp Syst* 2016; 17(12): 3434–3445.
57. Özgenel ÇF. (2019), "Concrete Crack Segmentation Dataset", *Mendeley Data*, V1, doi: 10.17632/jwsn7tfbrp.1
58. Liu Y, Yao J, Lu X, et al. DeepCrack: A deep hierarchical feature learning architecture for crack segmentation. *Neurocomputing* 2019; 338: 139–153.
59. Paszke A, Gross S, Chintala S, et al. *Automatic differentiation in pytorch*. Workshop Paper. 8. Neural Information Processing Systems (NIPS), 2017. https://openreview.net/forum?id = BJJsrmfCZ
60. Abraham N and Khan NM. A novel focal tversky loss function with improved attention u-net for lesion segmentation. In *2019 IEEE 16th international symposium on biomedical imaging (ISBI 2019)*, Venice, Italy, 2019, pp. 683–687. New York: IEEE.
61. Kingma DP and Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014. https://doi.org/10.48550/arXiv.1412.6980
62. Chen LC, Zhu Y, Papandreou G, et al. Encoder-decoder with atrous separable convolution for semantic image segmentation. In: Ferrari V, Hebert M, Sminchisescu C, et al. (eds.) *Proceedings of the European conference on computer vision (ECCV)*, Cham: Springer, 2018, pp.801–818.
63. Parrot Bebop. Parrot bebop 2 drone. https://www.parrot.com/us/support/documentation/bebop-range (2015, 20 June 2021).
64. Ackerman E. Parrot's new drone reclaims aniche: The Anafi marks the company's return to theconsumerspace-[Resources_Review]. *IEEE Spectr* 2018; 55(9): 21–21. https://www.parrot.com/us/drones/anafi/technical-specifications