International Journal of Micro Air Vehicles

Article

International Journal of Micro Air Vehicles
2018, Vol. 10(2) 171–185

DOI: 10.1177/1756829318757470
journals.sagepub.com/home/mav
SAGE

Deep learning for vision-based micro aerial vehicle autonomous landing

Leijian Yu[1], Cai Luo[2], Xingrui Yu[1], Xiangyuan Jiang[1], Erfu Yang[3], Chunbo Luo[4] and Peng Ren[1]

Abstract

Vision-based techniques are widely used in micro aerial vehicle autonomous landing systems. Existing vision-based autonomous landing schemes tend to detect specific landing landmarks by identifying their straightforward visual features such as shapes and colors. Though efficient to compute, these schemes only apply to landmarks with limited variability and require strict environmental conditions such as consistent lighting. To overcome these limitations, we propose an end-to-end landmark detection system based on a deep convolutional neural network, which not only easily scales up to a larger number of various landmarks but also exhibit robustness to different lighting conditions. Furthermore, we propose a separative implementation strategy which conducts convolutional neural network training and detection on different hardware platforms separately, i.e. a graphics processing unit work station and a micro aerial vehicle on-board system, subject to their specific implementation requirements. To evaluate the performance of our framework, we test it on synthesized scenarios and real-world videos captured by a quadrotor on-board camera. Experimental results validate that the proposed vision-based autonomous landing system is robust to landmark variability in different backgrounds and lighting situations.

Keywords

Micro aerial vehicle, vision-based autonomous landing, convolutional neural networks

Received 19 May 2017; accepted 11 January 2018

Introduction

In recent years, Unmanned Aerial Vehicles (UAVs) have been widely utilized in both military and civilian fields, such as military real-time monitoring, resource exploration, civil surveillance, cargo transportation and agricultural planning.[1] One key issue for safely applying UAVs to these tasks is to maneuver UAV flights in an accurate manner. Traditional UAV flights tend to be controlled through human manipulation with certain navigational aids. State-of-the-art UAV flights operate in an autonomous manner, which not only unleashes human labor but also enables safer and more accurate maneuvers. Specifically, three basic phases for UAV autonomous flights include takeoff, hovering and landing.[2] Among them, autonomous landing is the most crucial phase because 80% of the UAV accidents occur during landing.[3] Therefore, how to build robust autonomous landing systems has become one of the most important and challenging topics for the UAV research.[4]

Existing autonomous landing systems of UAVs can be roughly classified into two groups, i.e. electromagnetically guided landing systems and vision-based landing systems. The electromagnetically guided landing systems include those based on inertial navigation

[1]College of Information and Control Engineering, China University of Petroleum (East China), Qingdao, China
[2]College of Mechanical and Electronic Engineering, China University of Petroleum (East China), Qingdao, China
[3]Space Mechatronic Systems Technology Laboratory, Strathclyde Space Institute, Department of Design, Manufacture and Engineering Management, Glasgow, UK
[4]Department of Computer Science, College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter, UK

Corresponding author:
Peng Ren, College of Information and Control Engineering, China University of Petroleum (East China), 66 Changjiang West Road, Qingdao 266580, China.
Email: pengren@upc.edu.cn

systems (INS) and global positioning systems (GPS). INS provides instant positioning information but cannot guarantee long-term positioning accuracy. GPS provides a global availability in open areas but may incur positioning errors up to 10 m and may be blocked by buildings.[5] The electromagnetic landing systems are suitable for large scale landing problems (e.g. a large sized UAV landing on a large open area) with considerable tolerance for position errors. However, they cannot be straightforwardly applied to micro aerial vehicle (MAV) landing problems, which require accurate positioning within a small sized space. The electro-optical navigation can be considered as a transitioning landing technique between the electromagnetically guided landing and the vision-based landing, and generally serve as an auxiliary to electromagnetically guided landing systems. Vision-based landing systems use cameras to capture environmental visual features for the purpose of guided landing. One way to achieve this goal is to arrange cameras surrounding a landing area for capturing UAV/MAV status and environmental situations. One representative vision-based landing system in this regard is the VICON motion capture system. It is expensive and its application is limited to small indoor environments. In contrast to arranging off-boardMAV cameras like VICON, one more general configuration for a vision based landing system is to attach a camera on a MAV. By using optimal images of landing targets as source information for navigation, on-board vision systems can achieve positioning accuracy in terms of centimeters. This is especially valuable for MAVs that require more effective precise landing than larger sized UAVs. One on-board landing system identifies visual features of specific landing landmarks observed by the camera and accordingly guides MAV autonomous landing actions. In this scenario, specific landmarks are required to be designed as prerequisites for performing autonomous landing. In the literature, different specific landmarks are developed for different landing systems. Tsai et al.[6] designed the T-shaped landmark (Figure 1(a)) for their MAV autonomous landing systems. Saripalli et al.[7] designed the H-shaped landmark (Figure 1(b)) for their helicopter autonomous landing. Lin et al.[8] designed a landmark composed of eight equal-sized squares that are enclosed by a big white border (Figure 1(c)). Verbandt et al.[9] designed a landmark consisting of a series of concentric circles with exponentially distributed radii (Figure 1 (d)). Jung et al.[10] designed an H-shaped landmark with concentric circles (Figure 1(e)). These landmarks are designed to contain sharp or contrastive features that are easy to identify and segment from the background.

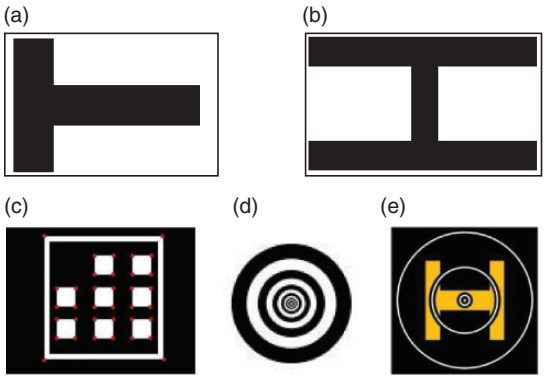

Figure 1. Samples of five widely used landmarks: (a) the T-shaped landmark; (b) the H-shaped landmark; (c) the landmark consisting of eight equal-sized squares and a big white border; (d) the landmark composed of a series of concentric circles; (e) the landmark composed of an H-shaped and concentric circles.

The key factor for vision-based landing systems is accurate landmark detection. Each existing vision based landing system is able to provide acceptable detection accuracy on its own landmark, but can hardly accurately detect landmarks for another system. This is because existing vision-based landing systems tend to detect specific landmarks through matching low level visual features such as shapes and colors between captured images and designed landmarks.[11] This case-by-case detection strategy is restricted to predefined landmarks and can hardly be generalized to a broad variety of landmarks. Furthermore, the visual quality of low level features extracted from captured images may also be easily devastated by inconsistent lighting conditions.

One intrinsic reason for these limitations is the machine learning techniques employed in the existing vision-based landing systems lack capability of learning high level visual features. In order to overcome these limitations, we exploit deep learning models for detecting the various landmarks under inconsistent lighting conditions. Deep learning is referred to a series of multilayer representational learning models that have attracted extensive research interests and achieved state-of-the-art performance on a number of artificial intelligence tasks.[12] In this paper, we describe how to exploit a deep convolutional neural network (CNN) model for detecting landmarks.

Traditional detection methods tend to first extract low level handcrafted features from captured images and then search the whole image for features that can match the predefined targets.[13,14] Recently, low level features are characterized in terms of region proposals, which are generated by feature learning techniques possibly being deep models. R-FCN[15] and MASK R-CNN[16] use region proposal networks to generate

detection proposals. Detection is then conducted via proposal classification. These methods consider feature extraction and feature matching as two separate steps and the speeds of these methods are slow because the networks in two stages are trained separately. In contrast to the two-step detection schemes, the end-to-end methodologies such as the Yolo methods use one network to predict the objects.[17,18]

Meanwhile, convolutional neural networks have been extensively studied in the deep learning literature. A number of attempts have been made for developing deeper and more complicated networks to achieve high accuracy.[19,20,21] However, these networks require extensive computational resources. On the other hand, resource limited platforms, such as the MAV on-board processors, are not qualified to implement these complicated networks.

Our framework is motivated by the recent proposed detection model Yolo[17] and the neural network architecture SqueezeNet[22] to achieve real-time landmark detection. The Yolo model frames object detection in terms of deep learning based regression for the purpose of determining spatially separated bounding boxes and associated class probabilities. The SqueezeNet aims at modeling a CNN with few parameters. In order to develop an effective end-to-end landmark detection system with implementation efficiency, we establish our CNN framework sharing advantages of the Yolo regression and the SqueezeNet efficient architecture. Specifically, our CNN-based landmark detection method regresses landmark positions directly from captured raw images through a multilayer architecture such that the feature extraction and matching are indistinguishably integrated into an overall framework. Furthermore, the strong representational power of the CNN not only increases the adaptability of an autonomous landing system from one specific landmark to multiple landmarks but also improves the detection robustness with respect to light variation.

Training our CNN based detection model is always time consumptive with heavy computational overheads. On the other hand, conducting detection based the trained CNN requires instant operations. To address these contradicted problems, we propose to train our CNN based detection model on a GPU workstation and operate the trained CNN model for detecting landmarks in the MAV on-board system. The separative implementations take advantages of both the GPU computational power and the on-board instant feedback, resulting in a novel strategy which leverages between comprehensively training and instantly operating deep models for MAV applications.

We experimentally test our CNN based landmark detection framework on synthesized scenarios and real-world videos captured by the on-board camera of a quadrotor. Experimental results validate that the proposed vision-based autonomous landing system is robust across various landmarks and different lighting situations.

Training a convolutional neural network for landmark detection

Inspired by the Yolo model[17] and SqueezeNet[22] modeling methodologies, we develop a convolutional neural network that performs end-to-end landmark detection. In this section, we first introduce the architecture for our convolutional neural network, and then describe how to train the CNN model for landmark detection on a GPU platform.

Convolutional neural network architecture for landmark detection

The convolutional neural network architecture of our proposed detection model is shown in Figure 2. Each input into the model is an RGB three channel image captured by an MAV on-board camera, and the corresponding outputs of the model are the predicted location of a detected landmark in the image and the predicted category label of the landmark. Specifically, one input image is first processed four convolutional and pooling layers, i.e. C_1, C_2, C_3, C_4, followed by one fully connected layer and one detection layer for regression. The blue cubes in Figure 2 indicate feature maps in each layer. Specifically, the C_{n-1} layer consists of K feature maps, i.e. $X_{n-1}^{(1)}, \cdots, X_{n-1}^{(K)}$, and these feature maps are the sources for computing the feature maps in the layer C_n.

To generate the lth feature map $X_n^{(l)}$ in the nth layer C_n, the feature maps in the $(n-1)$th layer C_{n-1} are processed by convolutional-activation-pooling (CAP) operations, which are basic operations in convolutional neural networks. Each feature map $X_{n-1}^{(k)}$ in the $(n-1)$th layer is convolved with learnable weights $W_n^{(k,l)}$. The sum of the K convolved results are added with learnable biases $b_n^{(l)}$, and further processed by a leaky rectified linear activation function $f(\cdot)$ which is depicted as follows:

$$f(m) = \begin{cases} m, & m > 0 \\ 0.1m, & m \leq 0 \end{cases} \tag{1}$$

The convolution-activation (CA) operations on the $(n-1)$th layer is formulated as follows:

$$\mathfrak{X}_n^{(l)} = f\left(\sum_{k=1}^{K} (W_n^{(k,l)} * X_{n-1}^{(k,l)}) + b_n^{(l)}\right) \tag{2}$$

where $*$ denotes the convolution operation.

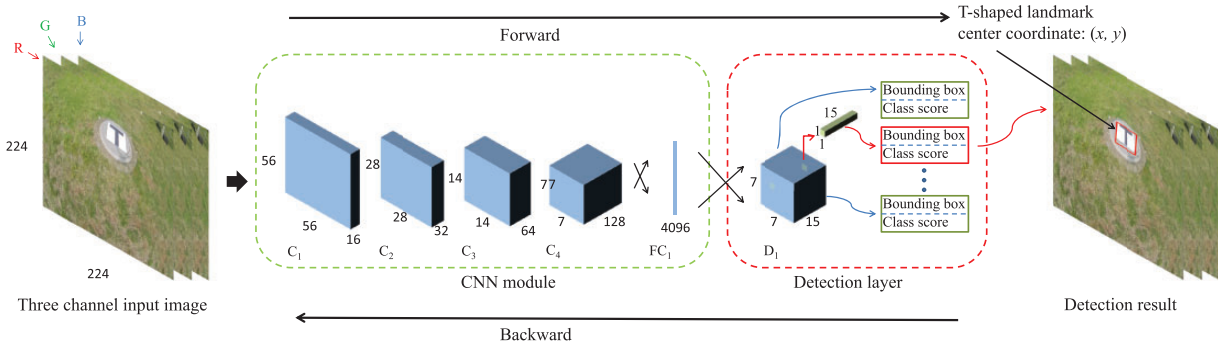

Figure 2. The full architecture of the proposed regression-based detection model. The architecture is composed of a CNN module and a detection layer. The input image is fed into the CNN module, followed by a detection layer. The detection layer provides the capability of regressing the coordinates and the class probabilities of the landmarks.
CNN: convolutional neural network.

$\mathfrak{X}_n^{(l)}$ is an intermediate feature map in C_n. It is further processed in terms of a pooling (P) operation for generating the feature map $X_n^{(l)}$ in C_n. Specifically, the pooling operation reduces the size of the $\mathfrak{X}_n^{(l)}$ by shrinking its 2×2 or 4×4 patches into single elements. This is done by replacing the patch by the largest valued element among the patch. The computation of the (i, j)th entry of the feature map $X_n^{(k,l)}$ is formulated as follows:

$$X_{n;i,j}^{(k,l)} = \max\{\mathfrak{X}_{n;i,j}^{(k,l)}, \mathfrak{X}_{n;i+1,j}^{(k,l)}, \mathfrak{X}_{n;i,j+1}^{(k,l)}, \mathfrak{X}_{n;i+1,j+1}^{(k,l)}\} \quad (3)$$

We describe the detailed configuration of our CNN landmark detection framework shown in Figure 2. In the layer C_1, the input three channel image is processed by a CAP operation and generate 16 feature maps of the size of 56×56. Similarly, C_2 has 32 feature maps of the size of 28×28, C_3 has 64 feature maps of the size of 14×14, and C_4 has 128 feature maps of the size 7×7. The detailed configuration is described in Table 1.

The parameter values 3×3 for the Conv Filter in C_1 refer to the size of each filter $W_1^{(k,l)}$ and the following parameter value 16 refers to the number K of filters applied in the layer. The parameter value 2 for Stride indicates the sliding step size for. The parameter values 2×2 for the Maxpooling indicate that pooling operations take place within a region of size 2×2. For C_2, C_3 and C_4, the parameter values have similar implications.

There are differences between the layers C_1 and C_4 and the layers C_2 and C_3. We design the layers C_1 and C_4 following Yolo.[17] On the other hand, different from Yolo, we design C_2 and C_3 by applying Conv Filters of the 'squeezed' size 1. This methodology is motivated by SqueezeNet[22] which replaces one big Conv Filter by parallel 'squeezed' Conv Layers and yields a simplified structure with reduced number of parameters. We exploit this advantage of SqueezeNet for conducting simplified CNN computation in an on-board system with limited computational resources. However, for on-board small CNNs, the SqueezeNet parallelism sacrifices certain accuracy for simplifying the CNN model. To remedy this ineffectiveness, we modify the parallelism into a serial implementation which is deeper and more effective to learn more complex feature representations.

Table 1. The CNN layer configuration.

C_n	Layer configuration
C_1	Conv Filter $3 \times 3 \times 16$, Stride 2
	Maxpooling 2×2, Stride 2
C_2	Conv Filter $1 \times 1 \times 8$, Stride 1
	Conv Filter $1 \times 1 \times 32$, Stride 1
	Conv Filter $3 \times 3 \times 32$, Stride 1
	Maxpooling 2×2, Stride 2
C_3	Conv Filter $1 \times 1 \times 16$, Stride 1
	Conv Filter $1 \times 1 \times 64$, Stride 1
	Conv Filter $3 \times 3 \times 64$, Stride 1
	Maxpooling 2×2, Stride 2
C_4	Conv Filter $3 \times 3 \times 128$, Stride 1
	Maxpooling 2×2, Stride 2

CNN: convolutional neural network.

The layer C_4 is followed by one fully-connected layer (FC_1) and then fully connected to a vector with 4096 dimensions. The full connection is depicted by cross arrows in Figure 2. Finally, the 4096 dimensional vector is processed by the detection layer (D_1) to generate a prediction tensor of the size $7 \times 7 \times 15$.

One prediction outputted by the CNN is represented as a 15 dimensional vector in the prediction tensor and the CNN generates 49 such prediction vectors for one input image. For each prediction vector, the first five entries represent the first prediction $(x_0, y_0, w_0, h_0, c_0)$, and the subsequent five entries represent the second

prediction $(x_1, y_1, w_1, h_1, c_1)$. Here (x_i, y_i) represent one predicted landmark centric coordinate and c_i reflects the confidence that a landmark is located within a grid cell surrounding (x_i, y_i) for $i \in \{1, 2\}$. Specifically, the confidence score is zero when no object falls into the grid cell. The confidence score is computed in terms of the intersection over union (IoU) between the predicted landmark and the ground truth as follows:

$$c_i \equiv \text{IoU} = \frac{\mathcal{A}_\text{O}}{\mathcal{A}_\text{U}} \tag{4}$$

where $\mathcal{A}_O$ and $\mathcal{A}_U$ denote the area of overlap and the area of union of the predicted landmark and the true landmark, respectively. We assume that there are totally five different categories of landmarks (as illustrated in Figure 1) used for landing, and the final five entries represent the probabilities $Pr(L_i), i = 1, \cdots, 5$ of the detected landmark belonging to one of the five candidate categories.

The class score s is computed by multiplying the conditional class probabilities and the individual confidence score for each landmark:

$$s = \Pr(L_i) * \text{IoU} \tag{5}$$

where Li denotes one of the landmark categories illustrated in Figure 1.

The best prediction is selected from the 49 predictions according to the highest class score s^*. For each prediction, the class score s is computed by equation (5). The best prediction consists of two components – the bounding box (x^*, y^*, w^*, h^*) and the class score s^*.

In the next subsection, we will comprehensively describe how to optimize the learnable parameters $W_n^{(k,l)}$ and b_n based on the prediction tensor.

Training the convolutional neural network on a GPU workstation

For training the CNN detection framework, we first resize the input image into 224 × 224 and then divide it into a 7 × 7 equally-sized grid cells. The cells are responsible for detecting the landmark if the center of the landmark falls into one of them. As described in the previous subsection, the CNN framework generates a $7 \times 7 \times 15$ prediction tensor for one input image, with each 15 dimensional vector in the tensor corresponding to one cell of the input image. The training procedure is to optimize the learnable parameters by minimizing the loss function measuring the differences between the prediction tensors and target tensors for input images.

The loss function consists of three parts, i.e. the area loss $\mathcal{L}_{area}$, categorical loss $\mathcal{L}_{cls}$, and the IoU loss $\mathcal{L}_{\text{IoU}}$, which are separately formulated as follows:

$$\begin{aligned}\mathcal{L}_{area} = {} & \lambda_c \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i,j}^{L} [(x_i - \widehat{x}_i)^2 + (y_i - \widehat{y}_i)^2] \\ & + \lambda_c \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i,j}^{L} [(\sqrt{w_i} - \sqrt{\widehat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\widehat{h}_i})^2] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i,j}^{L} (\mathcal{C}_i - \widehat{\mathcal{C}}_i)^2 \\ & + \lambda_{no} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i,j}^{no} (\mathcal{C}_i - \widehat{\mathcal{C}}_i)^2]\end{aligned} \tag{6}$$

where $\mathbb{I}_i^L$ denotes the confidence of landmark appearing in cell i, $\mathbb{I}_{i,j}^L$ denotes the jth (first or second) predictor in cell i is responsible for that prediction, and $\mathcal{C}_i$ indicates the class label of landmark in cell i. λ_c and λ_{no} are two balance parameters for making the training of the detection model more stable. In our design, we empirically set $\lambda_c = 5$ and $\lambda_{no} = 0.5$.

$$\mathcal{L}_{cls} = \sum_{i=0}^{S^2} \mathbb{I}_{i,j}^{L} (p_i(\mathcal{C}') - \widehat{p}_i(\mathcal{C}'))^2 \tag{7}$$

where $p_i(\mathcal{C}')$ is the conditional probability for landmark with label $\mathcal{C}'$.

$$\mathcal{L}_{\text{IoU}} = \sum_{i=0}^{S^2} \mathbb{I}_{i,j}^{L} (1 - c_i)^2 \tag{8}$$

where c_i is the confidence score, i.e. IoU, can be computed by equation (4).

The overall loss function is:

$$\mathcal{L} = \mathcal{L}_{area} + \mathcal{L}_{cls} + \mathcal{L}_{\text{IoU}} \tag{9}$$

Given a batch of training data (m image samples), we train the detection model via the stochastic gradient descent (SGD) algorithm. We first initialize the learnable weights W and biases b to a small random value near to zero subject to a normal distribution – *Norma* $l(0, \epsilon^2)$ with $\epsilon = 0.01$. During training, the input images are pushed forward (marked with the right-facing arrow in Figure 2) through the whole network to generate predictions. Then the errors in terms of the cost function equation (9) are measured. The error

gradients for the weights and biases are computed by equation (10) and backwardly propagated (marked with the left-facing arrow in Figure 2) for updating the parameter values.

$$\nabla_{W^{(l)}}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial W^{(l)}}. \qquad \nabla_{b^{(l)}}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial b^{(l)}}. \tag{10}$$

Algorithm 1: One iteration of stochastic gradient descent.

Set $\Delta W^{(l)} := 0$, $\Delta b^{(l)} := 0$

for $i = 1$ *to* m **do**

1. Compute the gradients $\nabla_{W^{(l)}}\mathcal{L}$ and $\nabla_{b^{(l)}}\mathcal{L}$ as equation (10).
2. Set $\Delta W^{(l)} = \Delta W^{(l)} + \nabla_{W^{(l)}}\mathcal{L}$.
3. Set $\Delta b^{(l)} = \Delta b^{(l)} + \nabla_{b^{(l)}}\mathcal{L}$.

Update the parameters:

$$W^{(l)} = W^{(l)} - \alpha\left[\left(\frac{1}{m}\Delta W^{(l)}\right) + \lambda W^{(l)}\right] \qquad b^{(l)} = b^{(l)} - \alpha\left[\frac{1}{m}\Delta b^{(l)}\right] \tag{11}$$

where α is the learning rate, λ denotes the weight decay parameter for adjusting the influence of model complexity, m is the number of images.

end

To train the CNN based detection model, we repeatedly take steps of the stochastic gradient descent as described in Algorithm 1 to minimize the loss function $\mathcal{L}$ in equation (9).

There are two things that need to be noted in our training procedure. First, the data jittering approach is employed to augment the landmark dataset. Specifically, the augmentation strategies operated on the landmark dataset include adjusting the image exposure, saturation and hue. The data augmentation increases the intraclass variability of training data and thus further improves the robustness of the detection model. Second, the training is carried out on a Graphics Processing Unit (GPU) work station, which is widely used for training deep learning models. However, it is not suitable for an MAV on-board system to perform CNN training by involving GPUs, which are normally physically big in size and comparatively power consumptive. On the other hand, it is not necessary to train a CNN in an on-board system because the MAV instant manipulation just requires implementing a trained CNN model rather than training it. In the light of this observation, we use an off-board GPU workstation for efficiently training our CNN framework. After the training procedure, we obtain about 4000 optimal parameters, i.e. optimal weights W^* and optimal bias b^*. We then transfer these trained parameter values to an on-board system for manipulating MAV landing, which is described in the next section.

Landmark detection based on the trained convolutional neural network

Landmark detection via CNN inference

The training procedure first forwardly computes the prediction tensor based the initial parameter values, and then adjusts the parameter values backwardly via back propagation optimization. In contrast to the forward-backward training procedure, the inference procedure processes each frame of a captured video through the CNN network only forwardly, based on the optimal parameter values (weights W^* and biases b^*). The forward computation generates the predicted coordinate (x^*, y^*) and class scores s^* for the detected landmark. Following the forward procedure (marked with theright-facing arrow in Figure 2), the inference procedure of an input three channels image I is summarized in Algorithm 2:

Algorithm 2: Landmark detection procedure with the optimal parameters W^* and b^*.

Set $L = 5$, $X_0^0 = I$, $Ks = \{3, 16, 32, 64, 128\}$

1. **for** $l = 1$ *to* L **do**
   (a) Set $K = Ks(l)$.
   (b) **for** $n = 1$ *to* 3 **do**
   Generate feature maps according to equation (2) with the optimal parameters

$$X_n^{(l)} = f\left(\sum_{k=1}^{K}\left(W_n^{*(k,l)} * X_{n-1}^{(k,l)}\right) + b_n^{*(l)}\right) \tag{12}$$

   end
   end
2. Process feature maps in the layer FC_1 based on W^* and b^*.
3. Process feature maps in the layer D_1 based on W^* and b^* to generate 49 predictions (15-dimension vector).

Each prediction can be presented as:

$$[(x_0, y_0, w_0, h_0, c_0), (x_1, y_1, w_1, h_1, c_1), (\Pr(L_1), \Pr(L_2), \Pr(L_3), \Pr(L_4), \Pr(L_5)] \tag{13}$$

4. Compute the class score s for each prediction according to equation (5).
5. Select the best coordinate prediction (x^*, y^*) according to the highest class score s^*.

Implementing detection on an MAV on-board system

We implement the detection model on the MAV on-board system Manifold, which consists of a quad-core ARM Cortex-A15 processor and 2 GB memory. The trained network is used for detecting landmarks from unlabeled video frames captured by the on-board camera. To run the inference procedure on Manifold, we build detection model with the 4000 optimal parameters (W^* and b^*) loaded on-board. Once the Manifold starts processing video captured by on-board camera, the inference procedure summarized in Algorithm 2 begins. The predictions in the form of $(x^*, y^*, w^*, h^*, s^*)$ is generated for guiding the landing.

It should be noted that we train and implement the CNN detection framework based on separate hardware platforms, i.e. the GPU workstation and the Manifold MAV on-board system. Though the CNN framework is developed with simplified architecture, it still requires considerable computational overheads especially in the training phase. Specifically, inferencing with the trained CNN is much less computational consumptive than training it, because the inference implementation does not involve the costly backward gradient computation. Therefore, in contrast to exploiting the computational power of the GPU workstation to handle the complex computation in the training phase, we implement the less complex detection procedure in the Manifold MAV on-board system, which is not only smaller in size, lighter in weight and less costive in power than the GPU workstation but also qualified to conduct an on-line real-time landmark detection.

Landing system

Figure 4 illustrates the operating procedures of the autonomous landing system. The image acquisition procedure is completed by capturing video frames using an on-board camera with universal serial bus (USB). The coordinates of the landmarks are predicted by forwarding the trained CNN detection model. The predictedcoordinates of the landmarks are then converted into x-axis and y-axis angular offsets (in radians) of the landmark center. These angular offsets are sent to the autopilot via the Micro Air Vehicles Communication Protocol (MAVLINK). This information is used to generate control signals to supervise the landing process via the autopilot. The detailed procedures are described in the following subsections.

Software architecture

The predicted landmark coordinates from videos captured by an MAV on-board camera are transformed to the MAV's own coordinates. We assume that the roll, pitch and yaw angles of the MAV can be neglected while computing the x-axis and y-axis angular offsets of the landmark coordinates in images. The x-axis and y-axis angular offsets are obtained as follows:

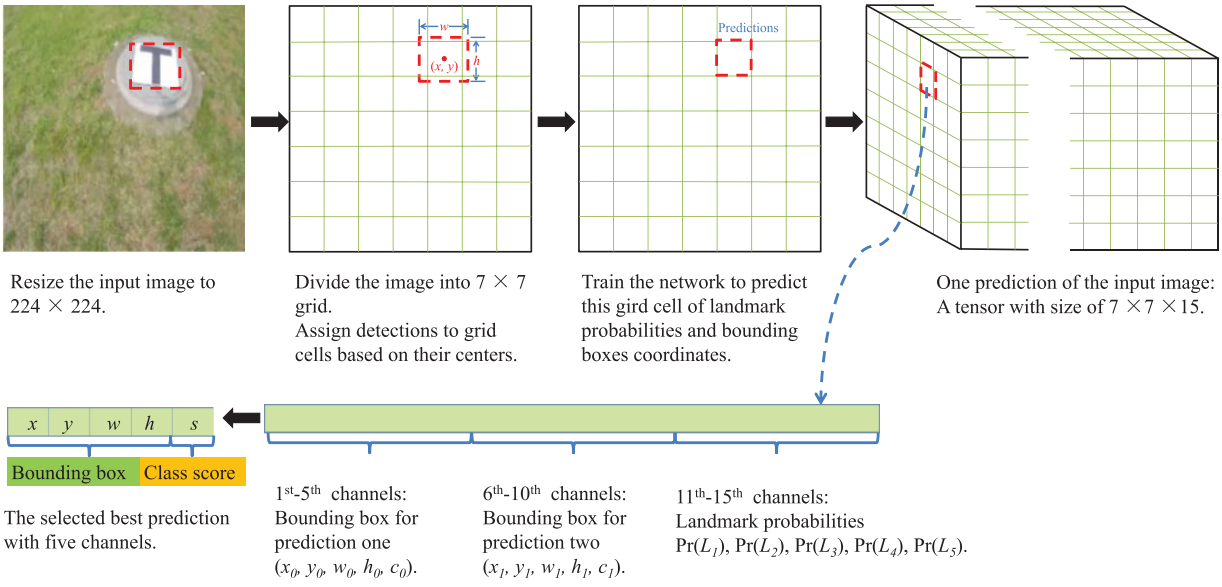

Figure 3. Detection procedure of the CNN-based detection model.
CNN: convolutional neural network.

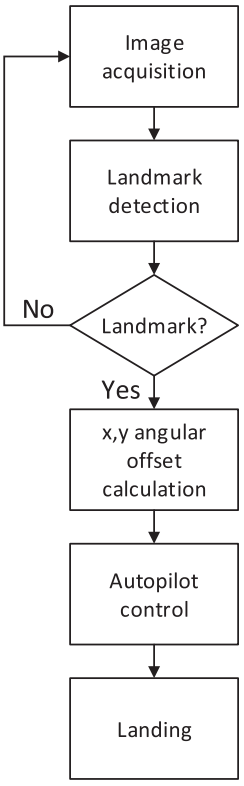

Figure 4. The flow chart of the vision-based MAV landing system.
MAV: micro aerial vehicle.

$$R_x = \frac{(x^* - hr/2) \times hf}{hr} \tag{14}$$

$$R_y = \frac{(y^* - vr/2) \times vf}{vr} \tag{15}$$

Here (x^*, y^*) denotes the coordinates detected by the trained detection model. *hr*, *hf*, *vr* and *vf* are on-board camera parameters. *hr* indicates the horizontal resolution, *hf* indicates the horizontal field of view (FoV), *vr* denotes the vertical resolution, and *vf* indicates the vertical FoV. Both *hf* and *vf* should be used in radians.

The R_x and R_y are sent to the autopilot as MAVLINK message. This information is processed with sonar data to compute landmark position relative to the MAV.

$$p_x = h_s \times \tan(R_x) \tag{16}$$

$$p_y = -h_s \times \tan(R_x) \tag{17}$$

where h_s is the height above the ground measured by sonar, and p_x and p_y are the x and y axis positions of the landmark relative to the MAV. The landmark velocities v_x and v_y in the x and y axis relative to the MAV can be calculated by doing differential operations with p_x and p_y.

A Kalman filter is then exploited for supervising the landmark, with the state vector defined as $\mathcal{X} = [p_x, p_y, v_x, v_y]^T$. The landmark equations are modeled as a linear system as follows:

$$\mathcal{X}_{k+1} = A\mathcal{X}_k + w_k \tag{18}$$

$$Z_k = H\mathcal{X}_k + u_k \tag{19}$$

where $\mathcal{X}_k$ is the true state vector describing the target position and velocity relative to the MAV at time k, A is the state transition parameter, w_k is the random process noise, Z_k is the measurement vector, H is the observation model, and u_k is the measurement noise. Let $p = [p_x, p_y]^T$ and $v = [v_x, v_y]^T$, and the motion can be characterized as follows:

$$p_{k+1} = p_k + v_k T_s + a_k T_s^2/2 \tag{20}$$

$$v_{k+1} = v_k + a_k T_s \tag{21}$$

where a_k is a random acceleration and T_s is the time step size. Furthermore, the Kalman state transition is:

$$\begin{bmatrix} p_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_k \\ v_k \end{bmatrix} + w_k \tag{22}$$

Assume that the modeling noise w_k is white zero-mean Gaussian noise with a covariance matrix Q, and the measurement noise u_k is white zero-mean Gaussian noise with a covariance matrix R:

$$p(w) \sim N(0, Q) \tag{23}$$

$$p(v) \sim N(0, R) \tag{24}$$

The state propagation and update equations for the Kalman filter are summarized as follows.

- The predicted state estimation is:

$$\widehat{\mathcal{X}}_{k|k-1} = A\widehat{\mathcal{X}}_{k-1} \tag{25}$$

- The predicted estimation covariance is:

$$P_{k|k-1} = AP_{k-1}A^T + Q \tag{26}$$

- The innovation covariance is:

$$S_k = HP_{k|k-1}H^T + R \tag{27}$$

- The optimal Kalman gain is:

$$\mathcal{K}_k = P_{k|k-1}H^T S_k^{-1} \tag{28}$$

- The updated state estimation is:

$$\widehat{\mathcal{X}}_k = \widehat{\mathcal{X}}_{k|k-1} + \mathcal{K}_k(Z_k - H\widehat{\mathcal{X}}_{k|k-1}) \tag{29}$$

- The posterior covariance is:

$$P_k = (I - \mathcal{K}_k H)P_{k|k-1} \tag{30}$$

In this group of equations, the superscript T indicates matrix transposition, $\widehat{\mathcal{X}}_{k|k-1}$ means the predicted state estimate value, $\widehat{\mathcal{X}}_{k-1}$ denotes the optimal state estimate value in the last step, $P_{k|k-1}$ indicates the covariance of the prediction error, $\mathcal{K}_k$ is the Kalman gain matrix, P_k denotes the covariance of the posterior estimate error, and $\widehat{\mathcal{X}}_k$ indicates the state estimate value.

The position and velocity estimate $\widehat{\mathcal{X}}_k$ of the landmark relative to the MAV are fed into the position controller of the autopilot, and the command is generated to manipulate the MAV to land on the landmark safely.

Hardware architecture

The hardware architecture is shown in Figure 5. In this system, we develop a customized do-it-yourself (DIY) quad-rotor with an embedded development board (Manifold). As the main processing unit, the Manifold board carries out the following tasks: (1) processing all images captured by the on-board camera; (2) calculating the angular offset; (3) communicating with the autopilot. The communication between the Manifold and the MAV main body is enabled by the Dronekit.

In our system, the Manifold connects to the UAV's autopilot through USB TTL Serial cables. The baud rate of the serial connection is 1,500,000, and the MAVLINK is used for communication. The Dronekit API is utilized as the development tool for our system.

The experimental quadrotor is shown in Figure 6, with a DJI NAZA F450 X-shaped frame and the Pixhawk autopilot with ArduPilot Firmware. Pixhawk integrates a 14 bit accelerometer, a 16 bit gyroscope, a magnetometer and an MS5611 barometer. A sonar is used to measure the height of the MAV above the ground. A USB camera is used to detect the landmark. The camera and sonar are configured as shown in Figure 7. The key component of the UAV's computing system is the Manifold, which consists of a quad-core ARM Cortex-A15 processor and 2 GB memory.

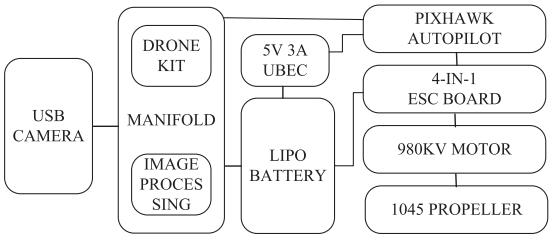

Figure 5. Hardware architecture of the vision-based autonomous landing system.
USB: universal serial bus.

Figure 6. The developed DIY quadrotor (from the top view of the MAV).
DIY: do-it-yourself; MAV: micro aerial vehicle.

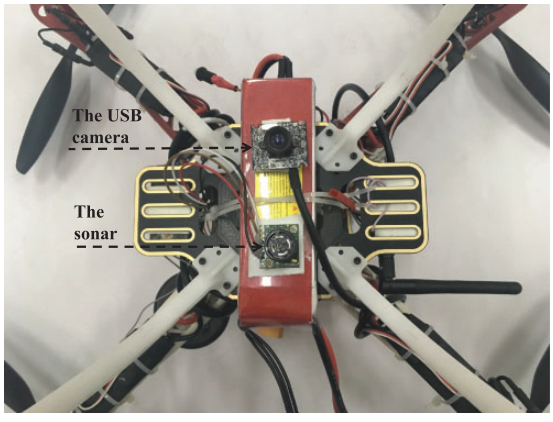

Figure 7. The configuration of camera and sonar (from the bottom view of the MAV).
MAV: micro aerial vehicle; USB: universal serial bus.

Experimental results

In the experiments, we use five categories of landmarks (as shown in Figure 1) for testing the performance of our framework. The landmarks size is 50 cm × 50 cm. The MAV with a downward looking camera is used to capture outdoor videos, and the flying height varies from 1 m to 5 m. We also use a forward looking camera for the MAV to capture indoor videos. The captured videos are separated into image frames. In order to train our model to be robust with respect to inconsistent lighting, we intentionally change image brightness and thus obtain various lighting training data. The brightness value ranges from 0 representing complete darkness to 100 representing complete whiteness. In our training process, the brightness values of 50 images from each landmark category are set to be 10, and those of another 50 images from each landmark category are set to be 90. We also use the data augmentation strategy presented in the previous section to enlarge the intraclass variability of the training data for the purpose of training a comprehensive model.

The images for each landmark category are annotated by using an open source tool – labelImg[1] https://github.com/tzutalin/labelImg. An obtained annotation file includes the category names and landmark coordinates. We build a training dataset containing images and their annotations. In our experiment, 200 images for each of the five landmark types are used to train our landmark detection model and the training data contains totally 1000 annotated images. The training process is realized on a workstation with a NVIDIA GTX860M GPU.

The values of IoU of our CNN model are shown in Figure 8. The IoU value reaches the peak after 35,000 iterations. We choose the CNN model with 35,000 iterations during the training procedure as our landmark detection model.

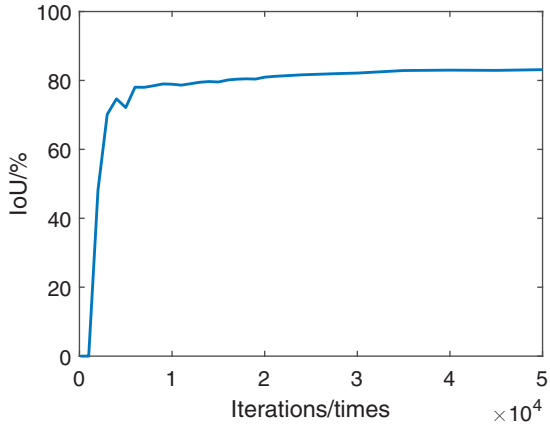

Figure 8. The values of IoU with different iterations. IoU: intersection over union.

In order to empirically evaluate the proposed onboard CNN-based landmark detection model, we use the test data sets, which are different from the training data sets, for testing our trained model. Specifically, we perform experiments on the Manifold board, which processes the outdoor and indoor real-time video frames as test data sets. Four tests are carried out:

- In the first set of experiments, we evaluate the robustness of the detection model for various landmark shapes.
- In the second set of experiments, we evaluate the robustness of the detection model for different illumination intensities.
- In the third set of experiments, we evaluate the robustness of the detection model for different background environments.
- In the fourth set of experiments, we evaluate the efficiency of the detection model.

Evaluation of the robustness of the detection model for various landmark shapes

To evaluate the effectiveness of our model for detecting various landmark shapes, we test our CNN model in terms of detecting each type of landmarks in 1000 images captured by the MAV camera. Results for the five landmarks are shown in Figure 9 and Table 2. Our CNN model successfully recognizes the landmarks in 4973 frames. The T-shaped landmark is mistaken for the H-shaped landmark in several rotation frames. This is because both the T-shaped landmark and the H-shaped landmark are simply featured by less discriminative black lines, and we only use 200 images of each type of landmarks to train our model. The landmark composed of an H-shaped and concentric circles is mistaken for the H-shaped landmark in several frames. We observe that these errors occur when the distance between the camera and the landmark is large. One reason for the misidentification of the H-shaped and circle landmarks is that they are symmetric shapes and are less distinguishable from distant views. To validate this observation, we use a fake target landmark, which appears similar to the circle landmark but is asymmetric, to replace the circle landmark for testing our model (Figure 9(f)). The experimental result reveals that our model does not misidentify the fake landmark as the true landmark. Our detection model can distinguish the landmark composed of a series of concentric circles and the fake targets from distant views correctly.

Qualitative evaluation results of the CNN model on the video frames captured from various flying heights and from different rotation angles are shown in Figure 10. The performance of our model is fairly stable when the MAV searching for the landmark at

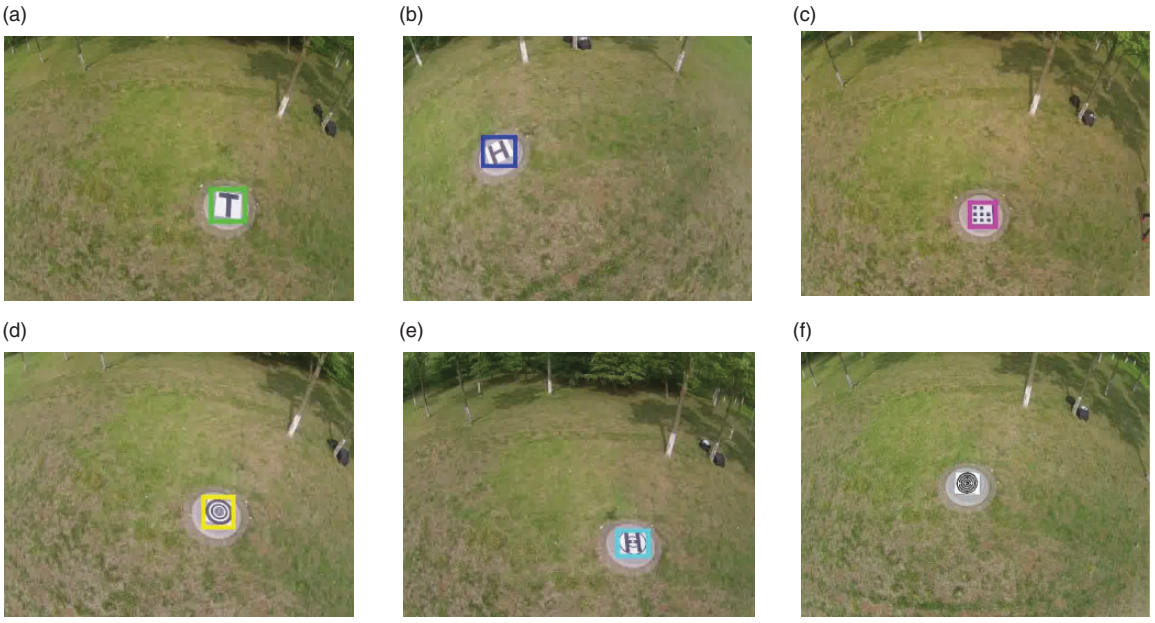

Figure 9. Landmark detection results: (a) the T-shaped landmark; (b) the H-shaped landmark; (c) the landmark consisting of eight equal-sized squares and a big white border; (d) the landmark composed of a series of concentric circles; (e) the landmark composed of an H-shaped and concentric circles; (f) the fake landmark.

Table 2. Results for evaluating the detection model for various landmark shapes.

	T-shaped	H-shaped	Square landmark	Circle landmark	Combined landmark
Number of test frames	1000	1000	1000	1000	1000
Correctly recognized frames	989	992	998	999	994
Detection accuracy	98.9%	99.2%	99.8%	99.9%	99.4%
Average time cost	47.53 ms	48.26 ms	47.35 ms	47.73 ms	48.62 ms

Figure 10. Detection results of the detection model on video frames captured from various flight heights ((a), (b) and (c)) and from different rotation angles ((d), (e) and (f)).

different heights and rotations. The processing rate is 21 frames per second.

To make the evaluation of the model robustness one step further, we use the MAV shown in Figure 6 to test the landing system with respect to different landmarks. The MAV takes off near the landmark to 4 m. When the landmark is detected, the vision-based landing system manipulates the MAV to autonomously land to the landmark region. All the actions in our test are performed by the MAV automatically. We test four flights for each landmark, and measure the horizontal distance between the MAV center to the landmark center in x and y axis. The position errors (e_x and e_y) are recorded in Table 3.

We can see from Table 3 that the position errors are acceptable for practical landing, because they are relatively small compared with the landmark size 50 cm × 50 cm. One reason for the position errors is that we assume the roll and pitch angles of the MAV to be zero during the MAV landing for the purpose of making condition controlled evaluations. This assumptive constraint causes some position measuring errors that do not arise from the detection model. Although suffering from these artificial errors, the MAV can still automatically land within the landmark region safely.

Table 3. Position errors.

	Mean	Variance	Max	Min
e_x	8.20 cm	14.88	14.35 cm	2.85 cm
e_y	9.11 cm	9.13	12.36 cm	4.05 cm

Evaluation of the robustness of the landmark detection model for different illumination intensities

In this set of experiments, 100 variously illuminated images for each landmark category are used for validation. Specifically, we generate different lighting situations by changing the brightness value of test images. The brightness values of the 100 images are set to be from 10 to 90 to test our landmark detection model. Visual results are shown in Figure 11. Experimental observations reveal that the detection results are stable for the MAV landmark navigation under various light conditions.

Evaluation of the robustness of the landmark detection model for different background environments.

To make the empirical evaluation of our model one step further, we test the detection performance of our method for detecting landmarks in different backgrounds. Specifically, we perform experiments on the videos captured in four different background

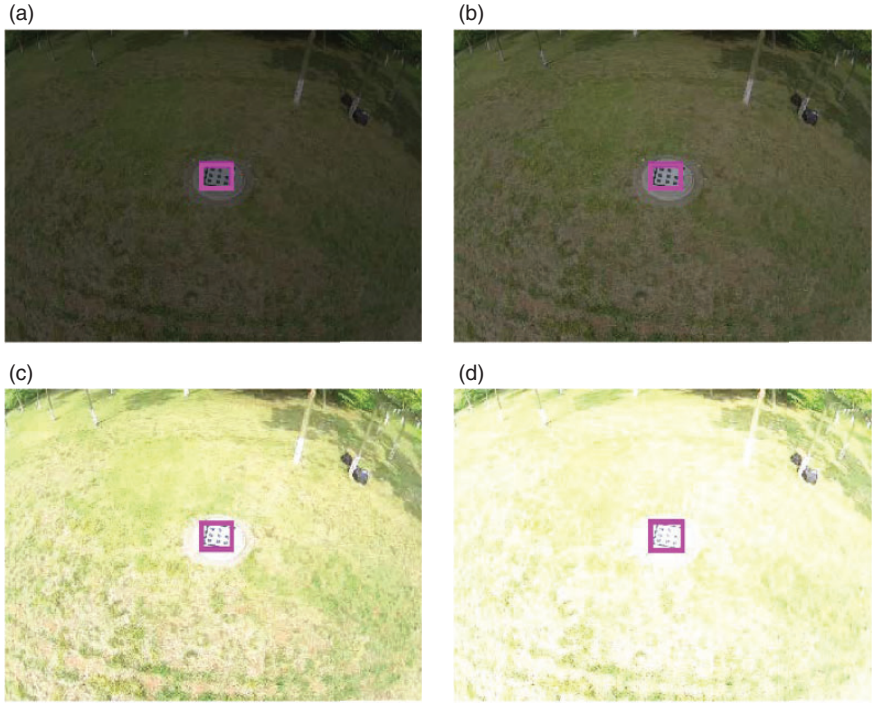

Figure 11. The detection results for the images with different brightness values: (a) the brightness value is 10; (b) the brightness value is 30; (c) the brightness value is 70; (d) the brightness value is 90.

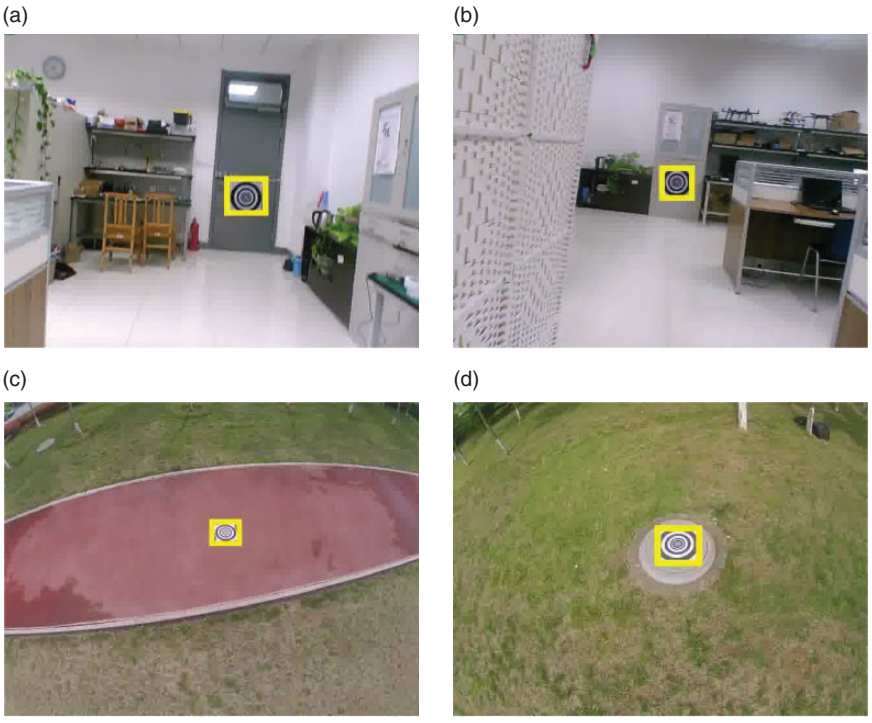

Figure 12. The landmark in various environments: (a) and (b) are indoor environments; (c) and (d) are outdoor environments.

Table 4. Results for evaluating the detection model for different background environments.

	T-shaped	H-shaped	Square landmark	Circle landmark	Combined landmark
Number of test frames	800	800	800	800	800
Correctly recognized frames	786	789	796	798	790
Detection accuracy	98.25%	98.625%	99.5%	99.75%	98.75%
Average time cost	47.62 ms	47.36 ms	47.25 ms	47.93 ms	48.32 ms

environments with the on-board camera. Two videos are captured outdoors, and the other two are captured indoors. We use 200 images captured in each environment to test our model. The results are shown in Figure 12 and Table 4. The experimental results validate that our landmark detection model is able to detect landmarks under various environments.

Evaluation of the efficiency of the landmark detection model

To make a quantitative comparison between our method and alternative state-of-the-art methods, we train the tiny models of Yolo[17] and Yolo v2[18] based on the same training dataset as that of our model. We then test the three trained models in terms of average IoU and processing rate (i.e. processed frames per second). The experiments are performed based on the same image frames as those in the first set of experiments. The comparison results are shown in Table 5.

Table 5. Comparison in terms of accuracy and efficiency.

	Yolo	Yolo v2	Our model
IoU	84.24%	89.29%	83.70%
Frames per second	7.5	5.3	21

IoU: intersection over union.

We observe that though the Yolo methods are slightly better in terms of accuracy, our model is much more efficient. The MAVs usually have limited computing resources on board (e.g. Manifold), which make the Yolo methods hardly achieve real-time implementation. On the other hand, our model achieves efficient implementation, which enables practical MAV onboard computation.

The usage of the CPU and memory the detection procedure is reported in Table 6. The landmark detection image processing algorithms use 3/4 of computing resources. Therefore, it allows more accurate control

Table 6. CPU and memory usage. CPU: central processing unit.

CPU1	CPU2	CPU3	CPU4	Memory usage
79%	74%	68%	68%	1407 MB/1892 MB

algorithms to execute during the landing procedure, which provides a possible route for improving the landing accuracy.

These experiments validate that our CNN model can not only detect the various landmarks, but also produce correct detections under different conditions. We put the detection results in video forms on the URL https://youtu.be/fifCK6BeDH8 for public observation. It is clear that our method is robust and efficient to process landmark information for guiding the MAV autonomous landing.

Conclusions

We have introduced a novel vision guided MAV autonomous landing system based on deep learning. Specifically, we have made three novel contributions. First, we have incorporated a modified SqueezeNet architecture into the Yolo scheme to develop a simplified CNN for detecting landmarks. Second, we have designed a separative implementation strategy which leverages the complex CNN training and the instant CNN detection. We have tested our novel framework in both synthesized and real-world environments and validated its effectiveness for MAV autonomous landing.

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: This work was supported by National Natural Science Foundation of China (Grant No.: 61671481 and 61701541), Shandong Provincial Natural Science Foundation(Grant No.: ZR2017QF003), Qingdao Applied Fundamental Research Project (Grant No.: 16-5-1-11-jch), the Royal Society of Edinburgh and National Natural Science Foundation of China joint project 2017-2019 (Grant No.: 6161101383) and the Fundamental Research Funds for Central Universities (Grant No.:15CX05042A and 16CX05004B).

References

1. Geng L, Zhang Y, Wang J, et al. Mission planning of autonomous UAVs for urban surveillance with evolutionary algorithms. In: *IEEE international conference on control and automation*, 2013, pp.828–833.
2. Yang S, Scherer SA and Zell A. An onboard monocular vision system for autonomous takeoff, hovering and landing of a micro aerial vehicle. *J Intell Robot Syst* 2013; 69: 499–515.
3. Tang D, Li F, Shen N, et al. UAV attitude and position estimation for vision-based landing. In: *International conference on electronic and mechanical engineering and information technology*, 2011, pp.4446–4450.
4. De Croon G, Ho H, De Wagter C, et al. Optic-flow based slope estimation for autonomous landing. *Int J Micro Air Vehicles* 2013; 5: 287–297.
5. De Croon G, De Clercq K, Ruijsink R, et al. Design, aerodynamics, and vision-based control of the delfly. *Int J Micro Air Vehicles* 2009; 1: 71–97.
6. Tsai AC, Gibbens PW and Stone RH. Terminal phase vision-based target recognition and 3D pose estimation for a tail-sitter, vertical takeoff and landing unmanned air vehicle. In: *Advances in image and video technology*. 2006, pp.672-681.
7. Saripalli S, Montgomery JF and Sukhatme G. Visually guided landing of an unmanned aerial vehicle. *IEEE Trans Robot and Autom* 2003; 19: 371–380.
8. Lin F, Chen BM and Tong HL. Vision aided motion estimation for unmanned helicopters in GPS denied environments. In: *Cybernetics and intelligent systems*. 2010, pp.64–69.
9. Verbandt M, Theys B and De Schutter J. Robust marker-tracking system for vision-based autonomous landing of vtol UAVs. In: *International micro air vehicle conference and competition*, 2014, pp.84–91.
10. Jung Y, Lee D and Bang H. Close-range vision navigation and guidance for rotary UAV autonomous landing. In: *IEEE international conference on automation science and engineering*, 2015, pp.342–347.
11. Fan Y, Haiqing S and Hong W. A vision-based algorithm for landing unmanned aerial vehicles. In: *International conference on computer science and software engineering*, 2008, pp.993–996.
12. Carrio A, Sampedro C, Rodriguez-Ramos A, et al. A review of deep learning methods and applications for unmanned aerial vehicles. *J Sensor* 2017;2: 1–13.
13. Felzenszwalb PF, Girshick RB, McAllester D, et al. Object detection with discriminatively trained part-based models. *IEEE Trans Pattern Anal Mach Intell* 2010; 32: 1627–1645.
14. Girshick R, Donahue J, Darrell T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation.In:*IEEE conference on computer vision and pattern recognition*, 2014, pp.580–587. *IEEE conference on computer vision and pattern recognition, 2014, pp.2014*.
15. Dai J, Li Y, He K, et al. R-FCN: object detection via region-based fully convolutional networks. In: *Advances in neural information processing systems*, 2016, pp.379–387.

16. He K, Gkioxari G, Dollár P, et al. Mask R-CNN. 2017, *arXiv preprint arXiv:1703.06870*.
17. Redmon J, Divvala S, Girshick R, et al. You only look once: unified, real-time object detection. In: *IEEE conference on computer vision and pattern recognition,* 2016, pp.779–788.
18. Redmon J and Farhadi A. Yolo9000: better, faster, stronger. 2016, *arXiv preprint arXiv:1612.08242*.
19. Simonyan K and Zisserman A. Very deep convolutional networks for large-scale image recognition. 2014, *arXiv preprint arXiv:1409.1556*.
20. He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition,* 2016, pp.770-778.
21. Szegedy C, Ioffe S, Vanhoucke V, et al. Inception-v4, inception-resnet and the impact of residual connections on learning. In: *AAAI*. 2017, pp.4278-4284.
22. Iandola FN, Han S, Moskewicz MW, et al. *Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size*, 2016, arXiv preprint arXiv:1602.07360.