Vision-based Perception with Safety Awareness for UAS Autonomous Landing

Zhenhao Zhao *, Jonathan Lee†, Zongyao Li‡, Chung Hyuk Park §, and Peng Wei¶
George Washington University, Washington, DC, 20052, USA

The use of small unmanned aircraft systems (UAS) has shown great potential for last-mile package delivery and medical supply transportation. In order to achieve higher levels of autonomy, scale up operations without tele-operating human pilots, and ensure landing safety during off-nominal events (e.g. people, pets, bikes, or cars being near/on the designated landing pad), we propose a robust, real-time deep learning based safe landing perception algorithm to (1) identify the landing pad, and (2) detect static or moving obstacles/humans near or on the landing pad. Specifically in this paper, we compare two state-of-art deep learning based computer vision models for object detection: RetinaNet and YOLOv5 to detect potential obstacles (pedestrians, cars and etc.). Additionally, we design and build a landing pad based on ArUco fiducial markers so we can detect the relative position and angle between the landing pad and the UAS with the ArUco library. Finally, we combined the landing pad and potential obstacle detection algorithms to ensure the landing pad is clear of obstacles. Our algorithm achieves real-time performance with 30 frame per second (FPS) video, suitable for real-world applications and further development.

I. Introduction

The use of small Unmanned Aircraft Systems (UAS) is promising for last-mile package delivery. Commercial small UASs have become more and more automated in executing pre-defined flight plans and landing on well-managed landing pads. However, to achieve higher level of autonomy during an off-nominal landing, a UAS needs to recognize landing scene reconfiguration such as moving or static obstacles near or on the designated landing pad. These obstacles often include people, pets, cars, and bikes. We explore the feasibility of using computer vision based perception to (1) identify the landing pad and (2) detect the obstacles near or on the landing pad. An accurate, fast and reliable perception module is a key component for a fully autonomous landing stack, together with a prediction module and planning/control module. In this paper, we focus on implementing and evaluating two state-of-the-art deep learning models for UAS perception, RetinaNet [1] and YOLOv5 [2]. We compare the two models' object detection accuracy and computation speed with real-world landing videos. Additionally, we evaluate the performance of the object detection on simulated landings on landing zones designated with Aruco markers.

Autonomous landing of UAS during off-nominal events has been a challenging research topic, which has been addressed with diverse approaches. Some of them compute the risk of collision with obstacle based on a UAV's predicted trajectory [3] and some of them try to explore the situation awareness issues [4] [5]. However, not much work has been done in applying deep learning based computer vision approaches to support small UAS autonomous landing during off-nominal events with real-time capabilities. Thus, this paper presents a major part of our research team's continuous effort on building a full-stack autonomous landing demonstration.

II. Related work

Our work extends the research of Yang et al [6]. The paper achieves a deep learning model trained for highly accurate multiple pedestrian detection. The paper identifies the application of real-time pedestrian detection to enhancing safety during the landing phase of a package delivery or emergency landing. We further develop this concept and determine the necessary algorithms to deploy a system on-board a small UAS.

*Graduate student, Department of Computer Science / Biomedical Engineering.
†Undergraduate student, Department of Computer Science.
‡Graduate student, Department of Computer Science.
§Associate Professor, Department of Biomedical Engineering.
¶Associate Professor, Department of Mechanical and Aerospace Engineering, AIAA Senior Member.

A system for safe landing zone identification via computer vision has been designed by Lusk et al [7]. The system, called Safe2Ditch, is an autonomous crash management system that reroutes the UAS to an alternative landing site if the presence of non-cooperative obstacles are detected in the current landing zone. Obstacle detection is achieved through recursive-RANSAC [8], a visual multiple target tracker, which can reliably identify moving obstacles from a moving camera. Whereas this approach was successful in determining the presence of a moving obstacle, it lacked the ability to classify objects and was not designed to detect stationary objects. Determining the type of object is valuable because it allows us the possibility to quantify levels of risk to humans, vehicles, or property. Thus, it remains important to utilize computer vision for both object localization and classification.

In the computer vision field, much attention has been spent developing computer vision based object detectors. The Histogram of Oriented Gradients (HOG) [9] and Deformable Part-based Model (DPM) [10] were the peak of traditional detection based models. In 2014, a category of models called two-stage detectors was pioneered by R-CNN [11]. Incremental performance improvements were made by other models such as Fast R-CNN [12] and Faster R-CNN [13]. This group of models features a distinct region proposal module and classification module, which remained a bottleneck for real-time applications. In 2015, Redmon et al. proposed a single-stage detector, YOLO [14] which greatly sped up detection time, but sacrificed in accuracy. Then RetinaNet introduced a new loss function Focal Loss which helped the model achieve high inference speed while outperforming two-stage detectors in terms of accuracy. Single stage detectors are still amongst the state-of-the-art, including the YOLO family of models. In this work we compare RetinaNet and YOLOv5 because they are lightweight models that will enable us to achieve real time performance, while still maintaining a high detection accuracy.

III. Methods

A. Dataset description and details

We trained the object detection models on the VisDrone2019-Det dataset [15], which is a public dataset collected by the AISKYEYE team at the Lab of Machine Learning and Data Mining, Tianjin University, China. The dataset consists of 7,019 static images captured by various drone platforms in 14 different cities throughout China. The images are taken from a variety of altitudes and camera angles, covering a broad distribution of locations from urban to rural with a variety of object densities. This makes the model trained on this dataset robust and very suitable for this study.

B. Deep learning model selection

We use a deep learning model to detect potential obstacles (cars, pedestrians, etc.). The model will return a list O, with each item in O being a potential obstacle:

$$[(x_{1i}, y_{1i}), (x_{2i}, y_{2i})]$$

The coordinates represent the top left corner and the bottom right corner respectively for the bounding box containing the obstacle, where i denotes the index of the obstacle.

In order to obtain higher accuracy and more real-time computing speed, we compared two object detection models that have excellent performances in other tasks: RetinaNet and YOLOv5.

1. RetinaNet

Architecture: The architecture of the RetinaNet is very similar to the Faster R-CNN[13]: It is mainly composed of the following three parts. Firstly, a backbone network for feature extraction. We selected ResNet50[16] to be the backbone. Secondly, a Feature Pyramid Network (FPN)[17] built on top of the backbone is used to integrate high level semantic information with low level semantic information. The FPN will improve the model's ability to identify multi-scale features. The final step is composed of two parallel fully connected subnetworks (subnets): (1) a classification subnet and (2) a box regression subnet. The subnets are used at each layer in the FPN to predict the probability of object presence at each spatial position, and regress the offset from each anchor box to a nearby ground-truth object respectively. Fig. 1 shows the structure of RetinaNet.

Focal loss: Traditional one-stage detectors, like YOLOv2 [14] [18] and SSD [19], struggle to deal with extreme foreground-background class imbalance. This is due to the presence of many candidate objects with different spatial locations, scales, and aspect ratios. Therefore, accuracy is greatly reduced compared with two-stage detectors. So a new loss function called Focal Loss was introduced to solve this problem: $FL(P_t) = -(1 - P_t)^{\gamma} \log(Pt)$. It adds

a factor $(1 - pt)^{\gamma}$ to the standard cross entropy formula. When γ is set to a positive value, the easy well-classified examples (background) will contribute less to the total loss function and the hard misclassified examples (foreground) will contribute more to the total loss function.

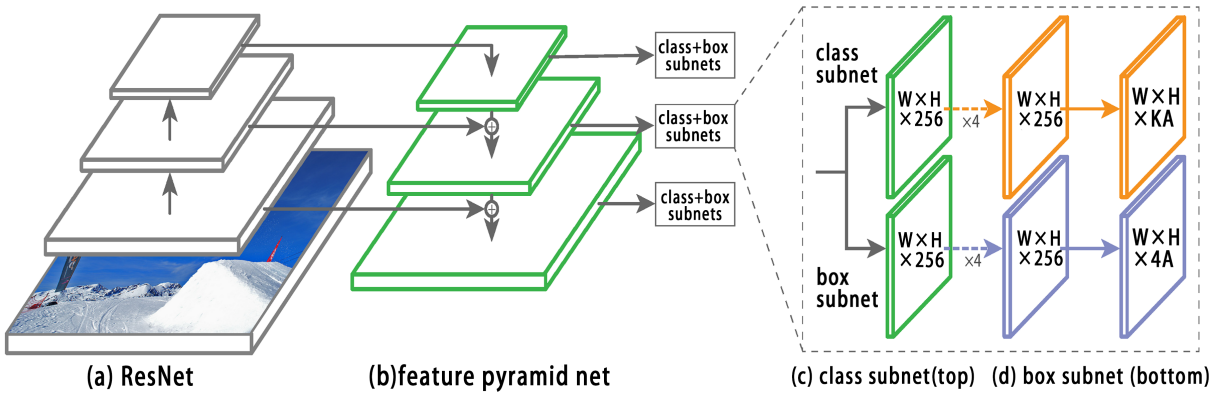

Fig. 1 The detailed structure for RetinaNet model.

2. YOLOv5

Architecturally, YOLOv5 is similar to YOLOv4 [20], but it adds a spatial pyramid pooling layer [21] and puts forward new data enhancement methods. YOLOv5 is composed of three key structures: the head, neck, and backbone. The backbone is a convolutional network to extract and process key features from the images. The neck uses the features from the backbones, with fully connected layers, to make predictions on probabilities and spatial information for the bounding box. The head is the final output layer of the network that generates anchor boxes for feature maps and outputs final output vectors with class probabilities and bounding boxes of detected objects. [2] The detailed structure of YOLOv5 has been shown in the Fig. 2.

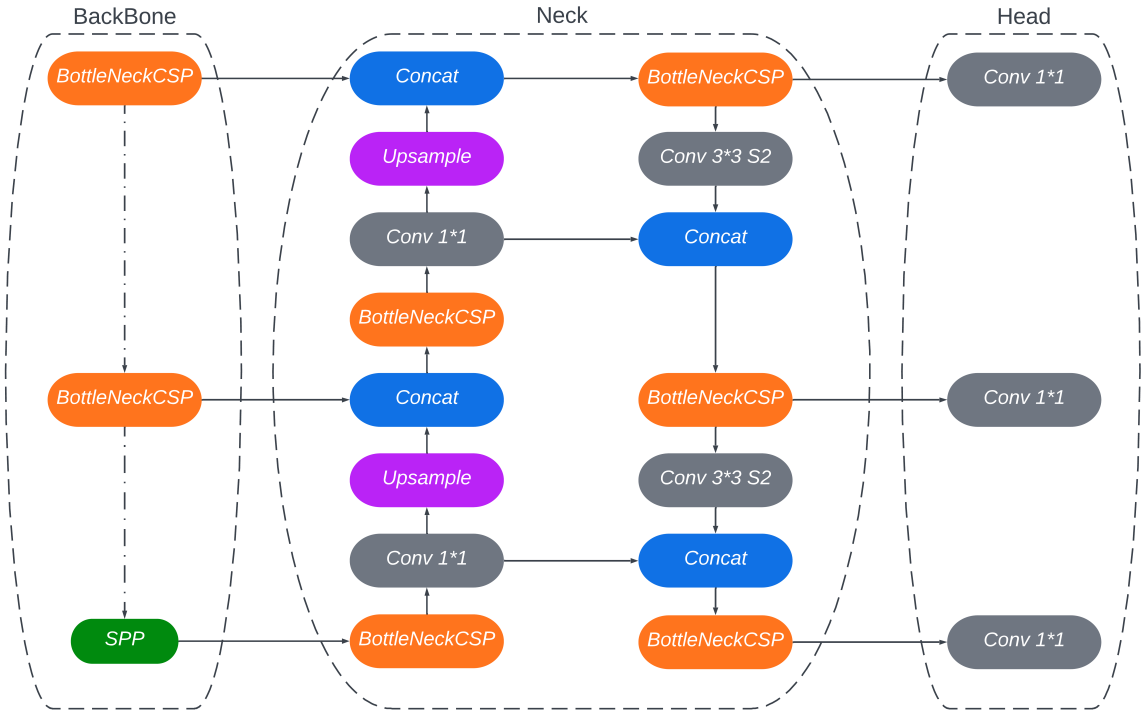

Fig. 2 The detailed structure for YOLOv5 model.

3. Comparison of YOLOv5 and RetinaNet

There are many technical differences between YOLOv5 and RetinaNet. However, the general structure remains similar: they both use a backbone network to extract the features of the original input. A neck structure is also used to combine and analyze the features extracted from the backbone network. Finally, a head structure is used to integrate the output of the neck and make the final detection. The focal loss first introduced in RetinaNet is also used in YOLOv5.

The key differences come from the technical details. First, the models use different backbone networks with the backbone of the YOLOv5 being more simple than RetinaNet. Compared to ResNet50, the two bottle neck CSP [22] structures of YOLOv5 have less layers and are more efficient. Next, the backbone of the YOLOv5 also uses a Spatial Pyramid Pooling layer [21] to balance the extraction of global features and the local features. Although the backbone of YOLOv5 is more simple and efficient, the backbone of RetinaNet is deeper and will catch more features. Additionally, the neck part is also slightly different, compared to the feature pyramid network used in RetinaNet. YOLOv5 adds a bottleneck CSP structure to further extract the features and, although this increases the computation cost, it is also helpful for increasing the detection accuracy. In the final output head structure, RetinaNet used two subnets to predict the classification and bounding box locations separately. Contrarily, YOLOv5 uses one network to directly output the classification and the bounding box locations.

Therefore, although YOLOv5 was created a few years after RetinaNet and used more advanced techniques, RetinaNet also has certain advantages in the comparison of some detailed structures. Hence, we chose to compare the accuracy and speed of both models to find the more efficient model for our landing safety detection algorithm to achieve high accuracy and fast inference speeds.

C. Landing pad with ArUco markers and its detection

In this study, We used ArUco markers[23] to build, detect, and track the landing pad. The ArUco library is based on OpenCV and enables the detection of various tag dictionaries. ArUco markers are synthetic square markers containing an inner binary matrix. There are several functions in the OpenCV library that can locate the markers precisely and quickly in the images and videos.

The landing pad consists of four ArUco markers placed on the upper left corner, upper right corner, lower left corner and lower right corner of the landing pad. Any ArUco marker tag can be used in each of the corners, as the landing pad does not have to face a particular orientation. Fig. 3 shows an example of the ArUco Marker and landing pad we built based on the 4 markers.

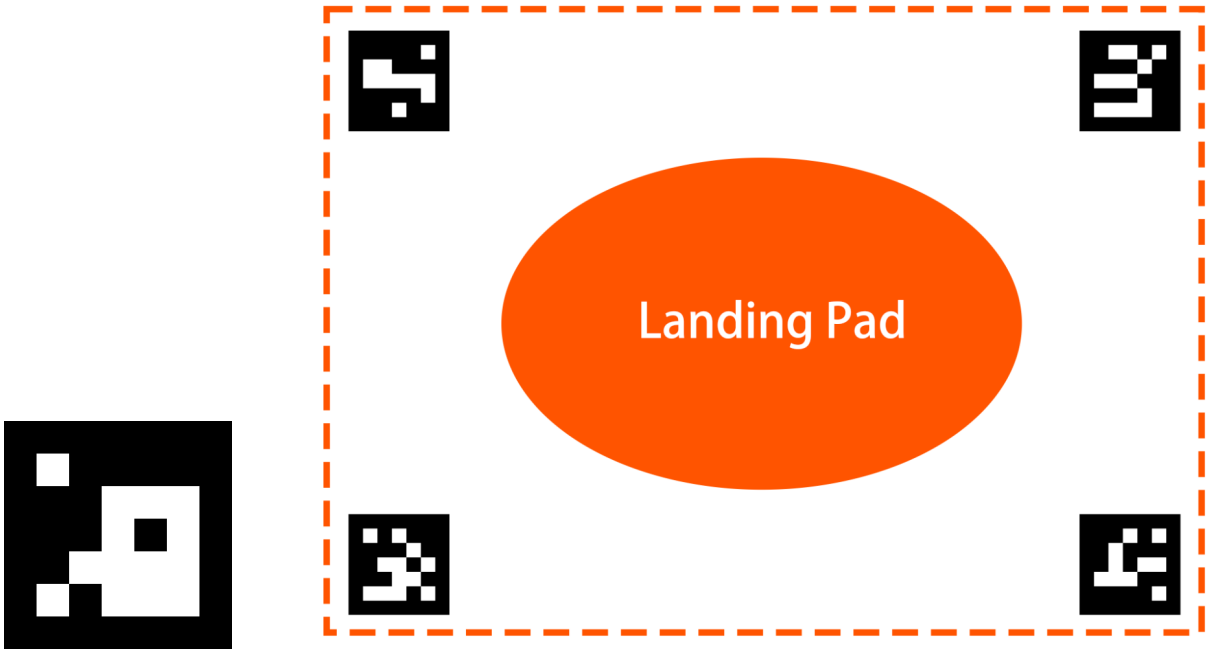

Fig. 3 Left: an example of the ArUco markers. Right: a landing pad built by 4 ArUco markers.

The landing pad detection algorithm is based on the ArUco marker detection. Each marker detection return four 2-D coordinates represent 4 corners of the marker respectively. Therefore, we can get a total of 16 coordinates for 4 markers. Then, we can store all the coordinate for x-axis in a vector $\vec{x}$ and y-axis in a vector $\vec{y}$. Both of $\vec{x}$ and $\vec{y}$ consist 16 elements. Finally, we can determine the coordinates of the landing pad: $(x_1, y_1), (x_1, y_2), (x_2, y_1)$ and (x_2, y_2) by

$$x_1 = min(\vec{x})$$
$$x_2 = max(\vec{x})$$
$$y_1 = min(\vec{y})$$
$$y_2 = max(\vec{y})$$

D. Landing pad safety checking algorithm

Above, we discussed how to use the deep learning models and the ArUco detection algorithm to obtain the bounding box of potential obstacles (pedestrians, vehicles, etc.) and landing pads in the video respectively. All bounding boxes have two pairs of xy-coordinates, which represent the coordinate of the top left corner and the bottom right corner. We will use this information to check if the landing pad is safe to land.

We first traverse obstacles list O to get two coordinates of one obstacle, $[(x_{1i}, y_{1i}), (x_{2i}, y_{2i})]$. And $(x_1, y_1), (x_2, y_2)$, which represent the two corners coordinates of landing pad. We then check if there is an overlap between the two areas checking the following conditions between the coordinates:

$$(x_1 \leq x_{1i} \leq x_2 \quad or \quad x_1 \leq x_{2i} \leq x_2) \quad and \quad (y_1 \leq y_{1i} \leq y_2 \quad or \quad y_1 \leq y_{2i} \leq y_2) \tag{1}$$

and

$$(x_{1i} \leq x_1 \leq x_{2i} \quad or \quad x_{1i} \leq x_2 \leq x_{2i}) \quad and \quad (y_{1i} < y_1 \leq y_{2i} \quad or \quad y_{1i} \leq y_2 \leq y_{2i}) \tag{2}$$

If Equation (1) or Equation (2) returns $True$, at least one of the obstacle bounding boxes overlaps the landing pad. Our algorithm will set a flag for the planning/control module to halt the landing. Algorithm 1 shows the whole process of our landing pad safety checking function.

Algorithm 1 Landing Pad Safety Checking

Require: IMG ← Image/one frame from the drone camera
  O ← Deep learning model(IMG)
  $(x_1, y_1), (x_2, y_2)$ ← Landing pad detection function(IMG)
  $alert \leftarrow FALSE$
  for each $[(x_{1i}, y_{1i}), (x_{2i}, y_{2i})] \in O$ **do**
    if $(x_1 \leq x_{1i} \leq x_2 \quad or \quad x_1 \leq x_{2i} \leq x_2) \quad and \quad (y_1 \leq y_{1i} \leq y_2 \quad or \quad y_1 \leq y_{2i} \leq y_2)$ **then**
      $alert \leftarrow TRUE$
    else if $(x_{1i} \leq x_1 \leq x_{2i} \quad or \quad x_{1i} \leq x_2 \leq x_{2i}) \quad and \quad (y_{1i} < y_1 \leq y_{2i} \quad or \quad y_{1i} \leq y_2 \leq y_{2i})$ **then**
      $alert \leftarrow TRUE$
    else
      continue
    end if
  end for
  return $alert$

IV. Results

A. Training and evaluating the deep learning models

We present experimental results on the VisDrone2019-Det dataset. We use a workstation running Ubuntu 20.04 with an Intel Xeon W-3245 @ 4.60 GHz (32 cores), 128GB RAM, and a RTX 3080 GPU (10GB) to train the models

and a Nvidia Jetson Xavier NX (Fig. 4) to evaluate the models. Additionally, we use a DJI Mini 2 to capture real-world landing footage on an experimental landing pad setup.

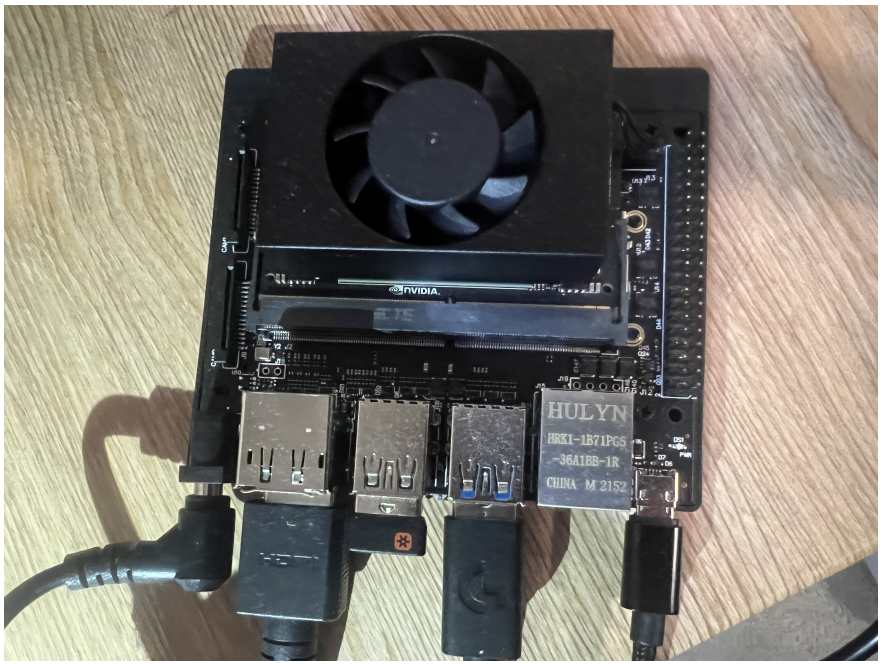

Fig. 4 The Jetson Xavier NX.

The RetinaNet and YOLOv5 models are pre-trained on the COCO dataset [24]. The training and validation images were normalized and scaled, and data was augmented by horizontal image flipping. All training was done with an adam optimizer and optimal hyperparameters were selected through a hyperparameter grid search. The grid search was conducted with cross validation to determine the best learning rate, batch size, and patience over 5 epochs. The patience parameter is part of the scheduler which decreases the learning rate if the loss has plateaued over a number of epochs. The best performing hyperparameters were then selected to train the model over 20 epochs. Table 1 shows the result of the grid search result. Fig. 5 and Fig.6 show the training process for the YOLOv5 model and RetinaNet. As the training progresses, the training and validation losses decrease, and model accuracy indicators increase. Mean Average Precision(mAP) is a metric used to evaluate object detection models. When the mAP is higher, the model should be more precise. The mAP of YOLOv5 is slightly higher than that of the RetinaNet with the best epoch.

Table 1 Hyperparameter grid search performance

Index	Learning rate	Batch size	Patience	Fold 0 mAP	Fold 1 mAP
1	0.001	4	1	0.0013	0.00075
2	0.001	4	3	0.0	0.0016
3	0.001	2	1	0.0023	0.0058
4	0.001	2	3	0.0	0.0016
5	0.00001	4	1	0.0168	0.0013
6	0.00001	4	3	0.0135	0.0013
7	0.00001	2	1	0.0138	0.0202
8	0.00001	2	3	0.0161	0.0203

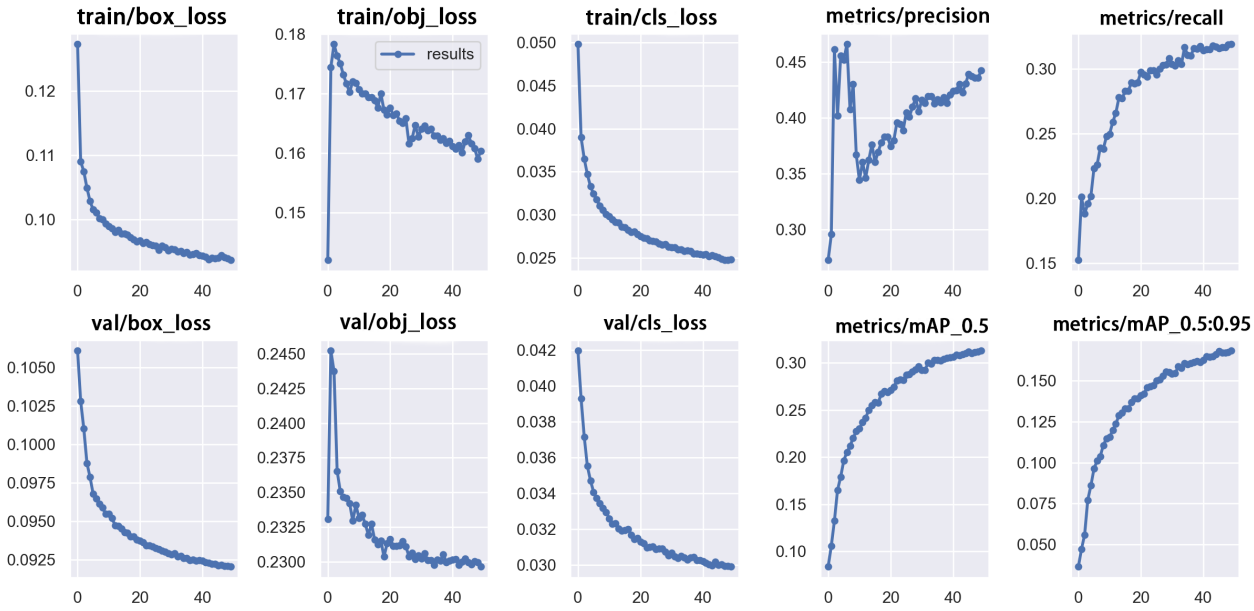

Fig. 5 The deep learning training plots for YOLOv5. The three columns on the left are the decreasing curves of various losses and the two columns on the right are the increasing curves of various accuracy indicators.

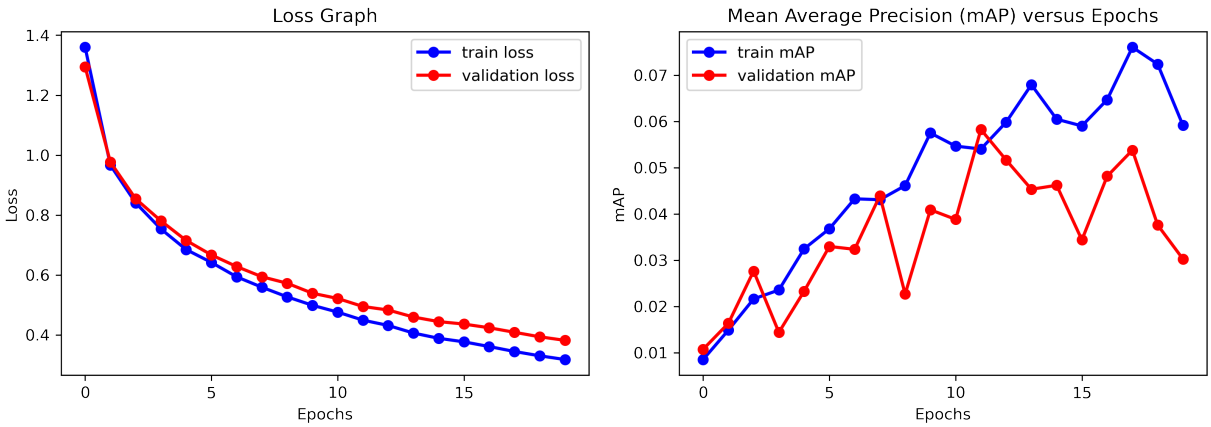

Fig. 6 Left: Loss on training and validation dataset. Right: Mean average precision on dataset during training and validation.

The confusion matrix (Fig. 7 and Fig. 8) shows predicted classes versus the ground truth. The result is as expected since most of the predictions are along the main diagonal. Note that there are many more pedestrians and cars than other classes due to the high class imbalance of the dataset. The model did misclassify other vehicle types as cars including vans, trucks, buses and motorcycles. We found that this is not a significant issue since all vehicles will be regarded as obstacles by the UAS.

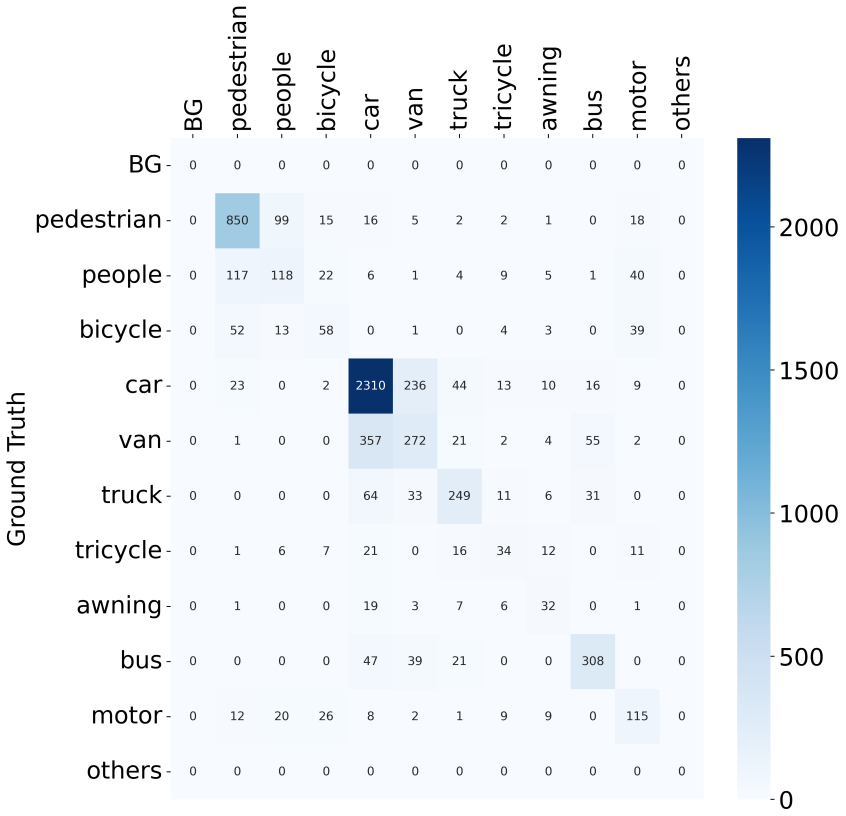

Fig. 7 RetinaNet model confusion matrix

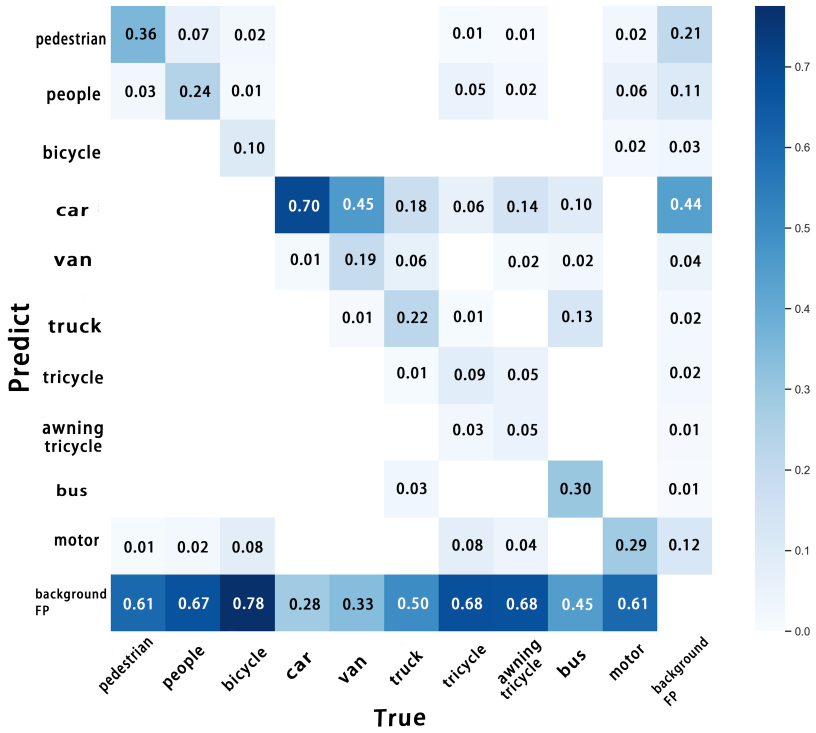

Fig. 8 YOLOv5 model confusion matrix.

Additionally, we tested the performance of the models on real landing videos taken on our DJI Mini 2 drone (Fig. 9 and Fig. 10). The Mini 2 captured 4K videos, on which we ran both computer vision models post-flight.

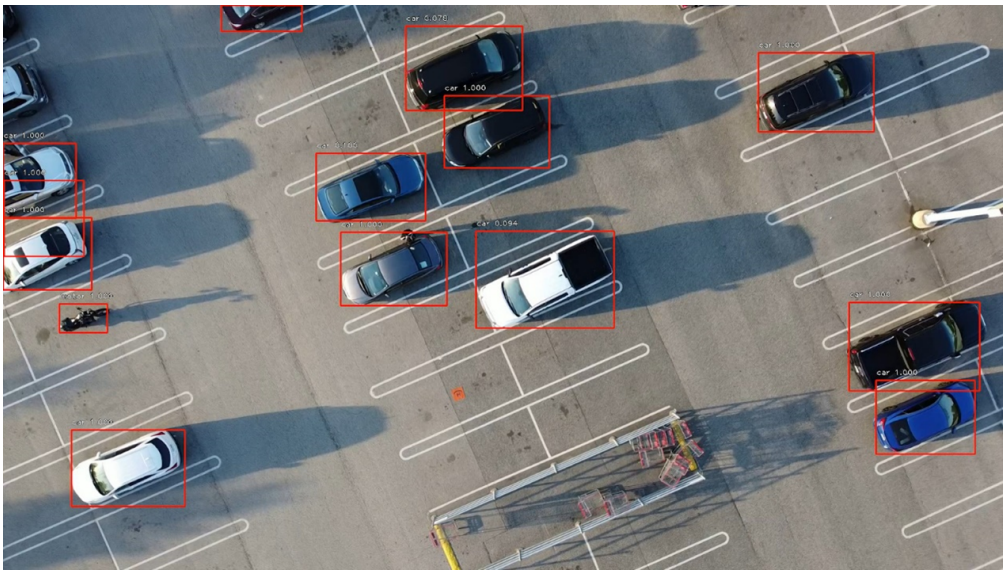

Fig. 9 RetinaNet model inference on one frame of the landing video.*

Fig. 10 YOLO model inference on one frame of the landing video.†

Table 2 shows the speed of the video inference for the two models on the different platforms. The inference speed of YOLOv5 is significantly better than that of RetinaNet. Based on the result, we decide to use YOLOv5 for our final safety landing algorithm.

*Full video: `https://www.youtube.com/watch?v=SGFQ4z6T_2o`.
†Full video: `https://www.youtube.com/watch?v=fQm7ZFBYHF4`.

Table 2 Comparison of model speed on a 14 second video on different hardware platforms

Model	Workstation:* seconds/frame	Workstation:* seconds/video	Onboard GPU:** seconds/frame	Onboard GPU:** seconds/video
RetinaNet	0.0476	20	0.7476	314
YOLOv5	0.0048	2	0.0619	26

* Intel Xeon W-3245 @ 4.60 GHz (32 cores), 128GB RAM, and a RTX 3080 GPU (10GB)
** Nvidia Jetson Xavier NX

B. Landing pad detection and safety checking

We built landing pad by ArUco marker and then test our landing pad detecting algorithm by self-collecting data (DJI Mini 2 drone). Fig. 11 shows the result.

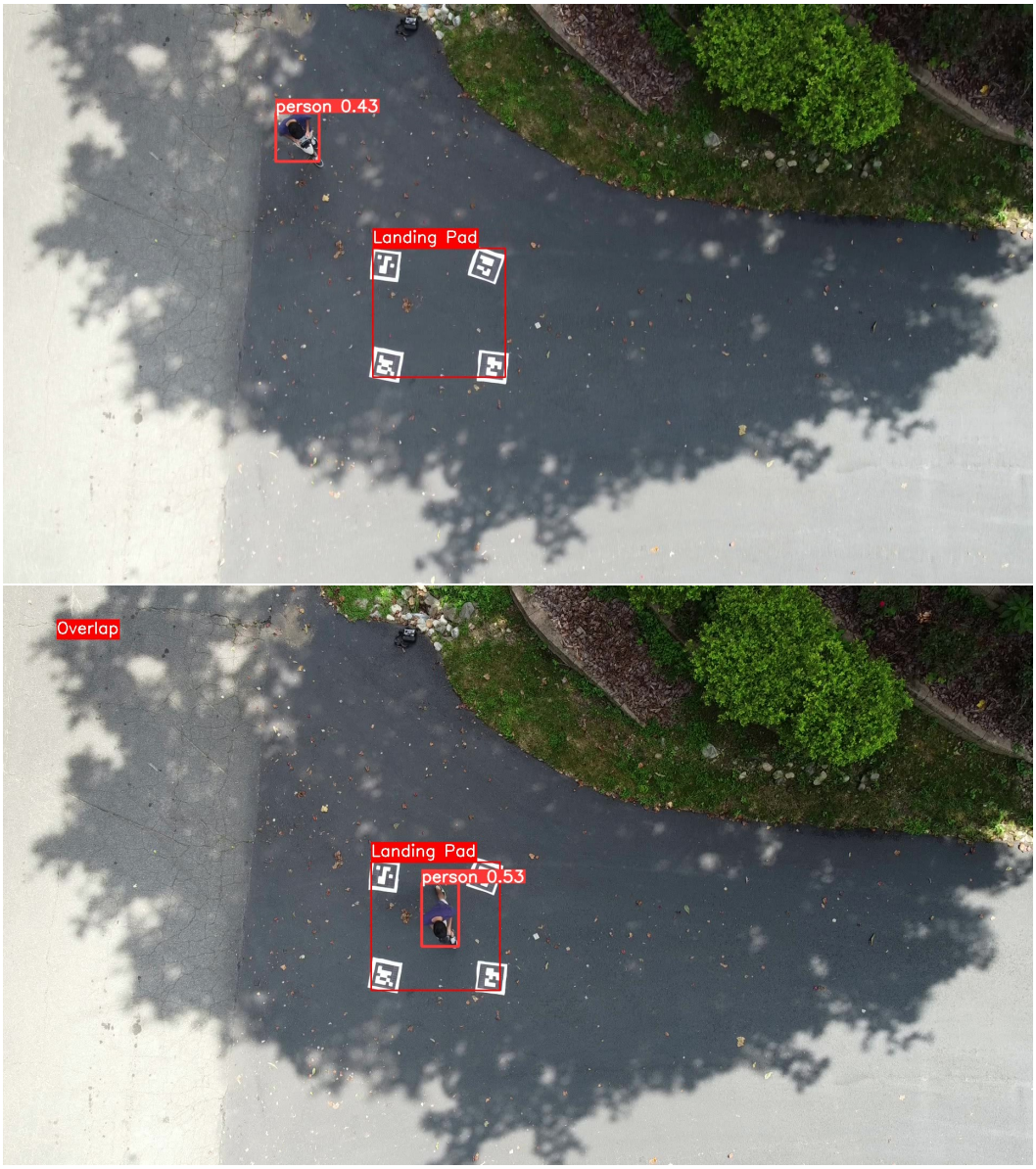

Fig. 11 The result of our algorithms. When a pedestrian walks into the target landing area from outside, the overlap is detected and an alert is given in the upper left of the video.*

*Full video: https://www.youtube.com/watch?v=aiQMhQI_wJE.

We ran our algorithm on the workstation (described above) and achieved an acceptable inference time. For a 53 seconds video, the total run time is 68.3656 seconds. This run time includes the algorithm initialization and visualization time. The average inference time per frame is only 0.0207s. The inference reached a maximum of 48.31 FPS which is higher than the video footage frame rate at 30 FPS. Therefore the algorithm meets our constraint for real-time performance.

V. Limitations

Our test scenarios were restricted in the environments available to perform tests. Due to safety concerns and flying restrictions, we could not perform tests in urban areas and in areas with many people. However, it would be desirable to test in a wider variety of scenarios that are representative of real-world landing situations. Additionally, we are limited to identifying object classes that are annotated in the VisDrone dataset. For example, our models would not be able to detect pets, such as cats or dogs, which may be present in potential residential landing areas. Additionally, our model was trained on a dataset collected only from cities in China, and generalization to detect vehicles and pedestrians around the world is also a concern. In the future, we can explore additional datasets and apply transfer learning to our models to address these problems.

In the videos we collected, our model was able to identify pedestrians and vehicles in the scene reliably from an altitude of between 2 and 20 meters. At altitudes above 20 meters, the landing pad markers appear too small to be detected. At altitudes less than 2 meters, it cannot be guaranteed that all four landing pad markers will appear in the view of the camera. We found this to be an acceptable range of altitudes, as the UAS could use a GPS and altimeter to approach the landing pad and descend to the desired height for landing pad and obstacle detection to be performed.

VI. Conclusion and Future Work

This paper proposed a landing pad safety detection algorithm using a computer vision-based approach. We first compared two state-of-art deep learning based object detection models for detecting potential obstacles in real-time. Next, we designed and built a landing pad constructed from ArUco markers. Finally, we applied our algorithm to self-collected data in a real-world environment. The model mAP reached above 0.32 and the average inference time per frame was 0.0207 seconds. The results show that our algorithm was able to successfully detected obstacles near/on a landing pad with YOLOv5 and achieved real-time detection.

We also demonstrated safety aware features by combining the vision-based landing pad tracking and obstacle (including pedestrians) tracking algorithm for safety landing. This was achieved in near real-time: 48.31 FPS. We understand this is only a beginning of the study for more safe and reliable UAS operations in the human environment, as the distance to the robot is only a basic feature in advanced human-robot interaction studies [25–27].

In future work, our object detection algorithm will be extended to consider predicted trajectories of obstacles and moving pedestrians as well as their behaviors. This would help identify potential hazards moving in a predicted trajectory towards the landing pad. Hence, the UAS could take action well before the obstacle enters the landing pad and alert the planning/control module to stop descending. Another direction of future research is to identify alternative landing areas using the detected bounding boxes information. This would make the landing process more flexible in the case of an emergency landing or presence of a stationary object in the landing zone.

Acknowledgement

We thank Pranav Gupta, our summer research intern from Langley High School, for collecting the test video in Figure 11.

References

[1] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollar, P., "Focal Loss for Dense Object Detection," *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[2] Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., NanoCode012, Kwon, Y., TaoXie, Fang, J., imyhxy, Michael, K., Lorna, V, A., Montes, D., Nadar, J., Laughing, tkianai, yxNONG, Skalski, P., Wang, Z., Hogan, A., Fati, C., Mammana, L., AlexWang1900, Patel, D., Yiwei, D., You, F., Hajek, J., Diaconu, L., and Minh, M. T., "ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference," , Feb. 2022. https://doi.org/10.5281/zenodo.6222936, URL https://doi.org/10.5281/zenodo.6222936.

[3] Banerjee, P., and Gorospe, G., "Risk assessment of obstacle collision for UAVs under off-nominal conditions," *Annual Conference of the PHM Society*, Vol. 12, No. 1, 2020, p. 9. https://doi.org/10.36001/phmconf.2020.v12i1.1194.

[4] Hunter, G., Wargo, C. A., and Blumer, T., "An investigation of UAS situational awareness in off-nominal events," *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, 2017, pp. 1–10. https://doi.org/10.1109/DASC.2017.8102038.

[5] Blumer, T., Wargo, C., and Hunter, G., "UAS situation awareness shortcomings, gaps, and future research needs," *2018 Integrated Communications, Navigation, Surveillance Conference (ICNS)*, 2018, pp. 2G1–1–2G1–8. https://doi.org/10.1109/ICNSURV.2018.8384850.

[6] Yang, X., Murphy, M., Brittain, M. W., and Wei, P., "Computer Vision for Small UAS Onboard Pedestrian Detection," *AIAA AVIATION 2020 FORUM*, 2020. https://doi.org/10.2514/6.2020-3270, URL https://arc.aiaa.org/doi/abs/10.2514/6.2020-3270.

[7] Lusk, P. C., Glaab, P. C., Glaab, L. J., and Beard, R. W., "Safe2Ditch: Emergency Landing for Small Unmanned Aircraft Systems," *Journal of Aerospace Information Systems*, Vol. 16, No. 8, 2019, pp. 327–339. https://doi.org/10.2514/1.I010706, URL https://doi.org/10.2514/1.I010706.

[8] Niedfeldt, P. C., Ingersoll, K., and Beard, R. W., "Comparison and Analysis of Recursive-RANSAC for Multiple Target Tracking," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 53, No. 1, 2017, pp. 461–476. https://doi.org/10.1109/TAES.2017.2650818.

[9] Dalal, N., and Triggs, B., "Histograms of oriented gradients for human detection," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Vol. 1, 2005, pp. 886–893 vol. 1. https://doi.org/10.1109/CVPR.2005.177.

[10] Felzenszwalb, P., McAllester, D., and Ramanan, D., "A discriminatively trained, multiscale, deformable part model," *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8. https://doi.org/10.1109/CVPR.2008.4587597.

[11] Girshick, R., Donahue, J., Darrell, T., and Malik, J., "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587. https://doi.org/10.1109/CVPR.2014.81.

[12] Girshick, R., "Fast R-CNN," *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, IEEE Computer Society, USA, 2015, p. 1440–1448. https://doi.org/10.1109/ICCV.2015.169, URL https://doi.org/10.1109/ICCV.2015.169.

[13] Ren, S., He, K., Girshick, R., and Sun, J., "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *Advances in Neural Information Processing Systems*, Vol. 28, edited by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf.

[14] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A., "You Only Look Once: Unified, Real-Time Object Detection," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[15] Zhu, P., Wen, L., Bian, X., Ling, H., and Hu, Q., "Vision Meets Drones: A Challenge," *CoRR*, Vol. abs/1804.07437, 2018. URL http://arxiv.org/abs/1804.07437.

[16] He, K., Zhang, X., Ren, S., and Sun, J., "Deep Residual Learning for Image Recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[17] Lin, T.-Y., Dollar, P., Girshick, R., He, K., Hariharan, B., and Belongie, S., "Feature Pyramid Networks for Object Detection," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[18] Redmon, J., and Farhadi, A., "YOLO9000: Better, Faster, Stronger," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[19] Fu, C., Liu, W., Ranga, A., Tyagi, A., and Berg, A. C., "DSSD : Deconvolutional Single Shot Detector," *CoRR*, Vol. abs/1701.06659, 2017. URL http://arxiv.org/abs/1701.06659.

[20] Yang, G., Feng, W., Jin, J., Lei, Q., Li, X., Gui, G., and Wang, W., "Face Mask Recognition System with YOLOV5 Based on Image Recognition," *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, 2020, pp. 1398–1404. https://doi.org/10.1109/ICCC51575.2020.9345042.

[21] He, K., Zhang, X., Ren, S., and Sun, J., "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 37, No. 9, 2015, pp. 1904–1916. https://doi.org/10.1109/TPAMI.2015.2389824.

[22] Wang, C.-Y., Liao, H.-Y. M., Wu, Y.-H., Chen, P.-Y., Hsieh, J.-W., and Yeh, I.-H., "CSPNet: A New Backbone That Can Enhance Learning Capability of CNN," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2020.

[23] Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., and Marín-Jiménez, M. J., "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognit.*, Vol. 47, No. 6, 2014, pp. 2280–2292.

[24] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L., "Microsoft COCO: Common Objects in Context," *Computer Vision – ECCV 2014*, edited by D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Springer International Publishing, Cham, 2014, pp. 740–755.

[25] Steinfeld, A., Fong, T., Kaber, D., Lewis, M., Scholtz, J., Schultz, A., and Goodrich, M., "Common metrics for human-robot interaction," *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, 2006, pp. 33–40.

[26] Mumm, J., and Mutlu, B., "Human-robot proxemics: physical and psychological distancing in human-robot interaction," *Proceedings of the 6th international conference on Human-robot interaction*, 2011, pp. 331–338.

[27] Burke, M., and Lasenby, J., "Pantomimic gestures for human–robot interaction," *IEEE Transactions on Robotics*, Vol. 31, No. 5, 2015, pp. 1225–1237.