IEEE Robotics and Automation Letters (RAL) paper presented at
2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

A High-Speed Detector For Low-Powered Devices In Aerial Grasping

Ashish Kumar†, Laxmidhar Behera†, *Senior Member, IEEE*

***Abstract*— Autonomous aerial harvesting is a highly complex problem because it requires numerous interdisciplinary algorithms to be executed on mini low-powered computing devices. Object detection is one such algorithm that is compute-hungry. In this context, we make the following contributions: (*i*) Fast Fruit Detector (FFD), a resource-efficient, single-stage, and postprocessing-free object detector based on our novel latent object representation (LOR) module, query assignment, and prediction strategy. FFD achieves 100FPS@FP32 precision on the latest 10W NVIDIA Jetson-NX embedded device while coexisting with other time-critical sub-systems such as control, grasping, SLAM, a major achievement of this work. (*ii*) a method to generate vast amounts of training data without exhaustive manual labelling of fruit images since they consist of a large number of instances, which increases the labelling cost and time. (*iii*) an open-source fruit detection dataset having plenty of very small-sized instances that are difficult to detect. Our exhaustive evaluations on our and MinneApple dataset show that FFD, being only a single-scale detector, is more accurate than many representative detectors, e.g. FFD is better than single-scale Faster-RCNN by 10.7AP, multi-scale Faster-RCNN by 2.3AP, and better than latest single-scale YOLO-v8 by 8AP and multi-scale YOLO-v8 by 0.3 while being considerably faster.**

Code & Dataset: https://github.com/ashishkumar822/FFD
Video: See attachment.

I. Introduction

Harvesting process in agriculture is a manpower-intensive and industrially important task, demanding high precision. With the rising applications of UAVs in agriculture, we foresee a huge scope of UAV-based grasping in the harvesting process. If one can harness the flying and maneuvering capabilities of UAVs, harvesting can continue 24×7 while significantly reducing the production costs, in outdoor orchards or recently emerged indoor vertical farming or precision agriculture.

However, developing a UAV-based fully autonomous harvesting system is not as straightforward as combining several algorithms and then deploying. It is because such a system should work in constrained and GPS-denied workspaces with entirely onboard computations, which in turn requires several algorithms/sub-systems to work in conjunction [1]. Such algorithms mainly include object detection, tracking, positioning system, control system, and grasping system, and running all of them at desired rates altogether on a low-powered, computationally limited device is a bottleneck. However, we believe that if each sub-system can be optimized as per the task requirements, the above issue can be resolved.

In harvesting automation, object detection is both crucial and a compute-intensive task. Although modern deep learning-based detectors offer high accuracy and parallelization, their high computational demands pose an issue for low-powered devices. It is so because other sub-systems also require a certain amount of computing to be run at desired rates. In addition, a high frame processing rate of the detector is also desired by the control system in order to perform visual servoing to accurately reach and grasp a target object [1].

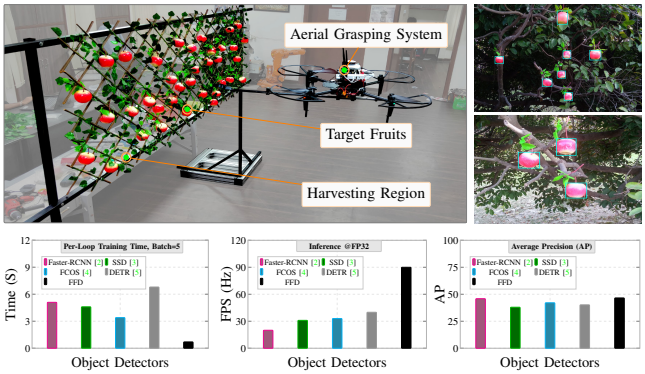

Fig. 1: Top: Our aerial grasping system for fruit harvesting, and outdoor detection. Bottom: FFD has low training time, high inference speed, and high detection accuracy compared to existing detectors.

Motivated by this, we translate the object detection problem into re-innovating the head of a detector, because it consists of most of the hand-tuned hyperparameters and time-consuming post-processing steps apart from the backbone. As a result, we propose *Fast-Fruit-Detector* (*FFD*) inspired by the hyperparameters and post-processing free design of recent Detection-Transformer (DETR) [5], while incorporating the task-centric observations from fruit harvesting, i.e. detection of fruits which appear smaller in images ($< 15 \times 15$ pixels).

FFD represents objects as queries which are obtained by our novel *Latent Object Representation* (LOR) module, directly from the backbone output instead of learning them [5]. These queries are used by our novel query assignment and matching strategy during the training phase. This turns FFD quite fast, accurate, resource-efficient, and postprocessing free while being a CNN-only design, free of compute-hungry Transformers. To the best of our knowledge, such speed and accuracy in the context of low-powered inference and robotic applications are still not visible in the literature. Despite we target FFD for fruits, it can be used in similar robotics applications. Summarily, main contributions of the paper are:

1) Single-stage and postprocessing free detector, achieving 100FPS@FP32 on 10W NVIDIA Jetson-NX (Sec. III).
2) A data multiplication approach to generate vast amounts of labelled training data from a small dataset (Sec. IV).
3) A challenging fruit detection dataset (Sec. V).

Next, we discuss related works, followed by FFD and the data multiplication approach. Experiments are described in Sec. VI, and Sec. VII provides conclusions on the paper.

†Department of Electrical Engineering, Indian Institute of Technology (IIT), Kanpur, India. {krashish, lbehera}@iitk.ac.in

II. Related Work

A. Convolutional Neural Network Based Detection

RCNN [6] fused traditional selective search for region proposal and CNN to obtain box and classification score. Fast-RCNN [7] proposed RoI-pooling to convert proposal features into a fixed size, thus improving both the speed and accuracy over RCNN. Then to avoid CPU-intensive and sluggish region proposal step, Faster-RCNN [2] proposed Region Proposal Network (RPN) and anchor boxes. RPN produces proposals as objectness score and coarse boxes relative to a huge number of anchors (~ 20000).

However, Faster-RCNN training becomes two-staged, complex, and has hand-crafted steps and hyperparameters to handle issues such as matching ground-truth boxes with a large number of anchors, class imbalance due to fewer positive anchors (object), and large negative anchors (non-object), positive-negative ratio for box-mining that consumes computing resources due to its CPU-only execution [7], [3]. This causes the accuracy and the runtime to be sensitive to the hyperparameter choices, thus necessitating hyperparameter tuning for a particular dataset which is a tedious process.

Further, the large number of anchor boxes produces high confidence for an object, resulting in redundant detections. NMS handles this issue via an intersection-over-union (IoU) threshold, however, it often discards small objects due to their low prediction confidence and as they occupy very small regions in the feature map. Therefore such objects are detected at high-resolution feature maps, but since these maps lack large context, multi-scale detection via feature fusion [8] is performed [2], [3]. It improves the accuracy but at the cost of increased run-time due to the processing of many anchors.

YOLO [9], SSD [3] speed-up the inference but at the cost of reduced accuracy by eliminating RPN, however, box-matching, postprocessing and multi-scale detection remain intact. FCOS [4] proposes an anchorless solution, however, postprocessing and feature fusion still exist. Moreover, mere backbone modifications [10], [11] or using depthwise separable convolutions in them [12] does not help, mainly because of the fundamental design limitations, i.e. anchor boxes, NMS, multi-stage detection which still remain in the picture.

The above limitations are bottlenecks in our case, i.e. the post-processing runtime overhead, and detecting small objects via FPN since fruits appear as small objects in the images.

B. Transformer Based Object Detector

Recent Detection-Transformer (DETR) [5] translates object detection into a set prediction problem while avoiding post-processing and hyperparameters entirely. DETR first encodes input image using a CNN, which is fed to a Transformer module, and then predicts a priori fixed number of objects via a Feed Forward Neural Network (FFN). DETR is simpler relative to the CNN-based detectors, however, its transformer blocks are a bottleneck for embedded computing devices both in terms of memory and computing resources. In addition, it suffers from slower convergence which limits its direct deployment in our case. Nevertheless, its design strongly motivates the development of the proposed detector FFD.

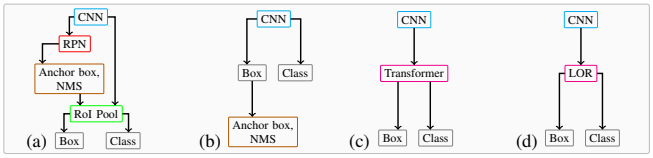

Fig. 2: (a) Faster-RCNN, (b) SSD, (c) DETR, and (d) FFD.

C. Detection For Fruit Harvesting

The application of object detection in agriculture automation is huge. [13] uses traditional feature-based vision for yield estimation. [14] uses Faster-RCNN for vegetable and fruit detection. [15] again uses Faster-RCNN for apple detection in orchards and mentions the importance of having a fast and accurate detector. [16] uses Gaussian-Mixture-Model (GMM) for counting and yield mapping in apple orchards.

Notably, these works employ existing detectors directly but do not focus on the detector design and improvements. As this is a fundamental requirement in this sector, we develop FFD for limited computing scenarios.

III. Fast Fruit Detector

Precisely, we aim to eliminate anchor boxes, NMS and multi-scale detection from a detector. The CNN-only detectors are architecturally simple, and converge faster but have complex training and testing steps, while DETR has simplified training and testing phases but is complex and converges slower [17]. Moreover, they are configured for large datasets [18] consisting of objects diverse in sizes, aspect ratio, and appearance, leaving room to incorporate task-centric observations when designing a detector. For instance, we target apple-like fruit which is quite small, and hence, efficient detection of small objects can be the main focus.

Since backbone is common among CNN detectors and DETR, with only differences in the prediction head strategy, we revisit both the designs, and re-innovate the detection head. This results in *FFD*, a single-staged, free of RPN, NMS or anchor-box detector having a simplified training and testing phase. Fig. 2 differentiates FFD architecture from the mainstream representative detectors.

A. Backbone

A large portion of the runtime is contributed by the backbone, therefore we choose VGG [19] network due to its plain structure and lower latency. We enhance it with BatchNorm [20] for faster convergence and better generalization. It is five staged with $\{2,2,3,3,4\}$ layers and $\{16,32,64,128,256\}$ neurons per stage, each operating at a stride of 2, and the final one producing a tensor $T_f \in \mathbb{R}^{C\times H_o\times W_o}$, where $C = 256$, $H_o = \frac{H}{32}$, and $W_o = \frac{W_o}{32}$, H, W are the image height and width.

As small objects lose their identity in low-resolution feature maps, multi-scale detection [8] is employed. However, we aim to detect them only from low-resolution map T_f to reduce computational complexity (Sec. II-A). To achieve that, we propose a latent object representation (`LOR`) module that is motivated by the query-key-value paradigm of [5] but is free of transformer attention mechanism and is fully convolutional.

Note: Backbone can be chosen to be any other network depending on the difficulty of a dataset and desired accuracy.

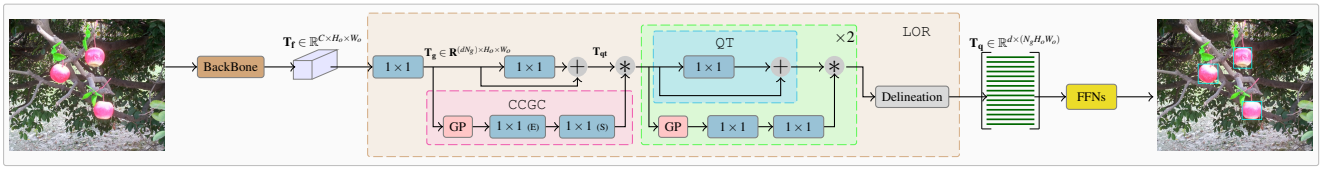

Fig. 3: Fast-Fruit-Detector (FFD). "GP": Global Pooling,'E': Expand, 'S': Squeeze, and '⊛' Broadcast multiplication.

B. Latent Object Representation (LOR)

Here we refer the reader to DETR concepts [5] to better understand the upcoming text. DETR produces a fixed number of queries, each representing an object. The queries are initialized via embeddings [5] or anchor-boxes [17], and iteratively refined via compute-intensive self-attention and cross-attention of transformer encoder and decoder blocks. Regardless of the query design (embedding or anchor-box), they are not generated from the backbone in any of [5], [17].

On the contrary, we propose to generate them via LOR module (Fig. 3) directly from the backbone output in a computationally efficient manner, without transformers. This results in an extremely simplified detection pipeline which is also free of post-processing. To the best of our knowledge, this query design is novel and FFD is the first to utilize it.

The LOR module can be divided into two parts: query transformation (QT), and cross-channel global context (CCGC).

1) Query Transformation QT: In this step, the input tensor (T_i) to the LOR module is passed through a 1×1 convolution whose output is added to the input (residual connection [21]). QT essentially adds non-linearity to the input queries and results in a tensor T_{qt}, denoted as below:

$$T_{qt} = \text{ReLU}(\mathscr{F}_{qt}(T_i) + T_i), \quad \mathscr{F}_{qt} \equiv \text{Conv}_{1\times1} \tag{1}$$

2) Cross Channel Global Context (CCGC): The output of the backbone (T_f) is devoid of large spatial context due to the shallow backbone that limits its receptive field. However, the role of contextual information in detection and segmentation is crucial [22]. Although there are many ways [22] to do so, we devise a simple and compute efficient strategy CCGC.

In this strategy, we pass the input through global pooling, producing a 1D tensor $\mathbf{z} \in \mathbb{R}^C$ whose i^{th} channel is given by:

$$z_i = \frac{1}{H_o \times W_o} \sum_{h \in H_o, w \in W_o} T_i(h,w) \tag{2}$$

where, H_o, W_o are the height, and width of the input tensor.

At this point, elements of $\mathbf{z}$ carry global context but lack cross-channel context. Thus to embed the cross-channel context, $\mathbf{z}$ is transformed via two sequentially connected convolution layers which intertwine the content of z_i's; the first layer expands the input channels by a factor r ($\mathscr{F}_e$) while the other squeezes them by the same factor ($\mathscr{F}_s$), denoted as:

$$\mathscr{F}_e \equiv \text{ReLU}(\text{Conv}_{1\times1}), \quad \mathscr{F}_s \equiv \sigma(\text{Conv}_{1\times1}) \tag{3}$$

where, $\sigma(\cdot)$ stands for Sigmoidal activation.

A similar structure with additional operations is employed in [23] but is intended to improve CNN's accuracy. On the contrary, our use is entirely different i.e. aggregating global information in a simplified possible manner.

The resulting tensor is now broadcast multiplied [24] with T_{qt} which weights T_{qt} information depending on the global context. Summarily, CCGC adds non-linearity to $\mathbf{z}$ which is propagated to T_q by amplifying salient information in T_{qt} through broadcast multiplication. CCGC can be written as:

$$\mathscr{F}_{\texttt{ccgc}} \equiv \mathscr{F}_{\text{s}}(\mathscr{F}_{\text{e}}(\text{z})), \tag{4}$$

Overall Flow: LOR module takes input the tensor T_f which is operated upon by a 1×1 convolution, producing a tensor T_g of channels dN_g, where d is query dimension, and N_g queries exist per spatial location $\in \mathbb{R}^{H_o \times W_o}$ of T_f.

Now QT and CCGC modules are used in parallel and repeated three times to learn better data representation, while still having access to a wider spatial context. Adding more of such modules increases parameters but does not add to accuracy, because the backbone is still fixed. We perform repetition only three times to meet our runtime requirements, however, they are flexible enough to be adjusted. The overall flow of LOR is shown in Fig. 3 and is summarized as follows.

$$\mathscr{F}_{\texttt{LOR}} \equiv \bigodot_{i=1}^{N} T_{qt} * \mathscr{F}_{\texttt{ccgc}} \tag{5}$$

where, $\odot$ is function-of-function, $*$ is broadcast multiplication.

C. Delineation

The output of LOR is now collapsed spatially, resulting in *query matrix* $T_q \in \mathbb{R}^{d \times (N_g H_o W_o)}$, whose each row denotes a query $\mathbf{q}$ that represents an object detectable in the image.

In LOR, queries in the form of learnable embeddings [5] or anchor-box [25], [17] are not needed, instead they are directly generated from the backbone output. This is the major novelty of the LOR module, leading to a simplified structure, high accuracy without needing post-processing, and faster speeds.

D. Prediction

The tensor T_q is forwarded to two Feed Forward Networks (FFN) which are a stack of 1×1 convolutions followed by ReLU [5]; One for Classification (FFN_c) having one layer, and one for box regression (FFN_b), having three layers.

E. Query Assignment

DETR predicts w.r.t. the image origin $(0,0)$, whereas [17], [25] predicts w.r.t. the learned anchors. It limits the total number of detectable objects in the image, regardless of the image resolution. To handle that, we propose to generate N_g queries per spatial location of T_f, and each such location refers to a non-overlapping tile of the input image following [26]. With this strategy, each set of N_g queries in T_q corresponds to all the objects whose center lies in a particular tile (Fig. 4). It is the uniqueness of FFD queries in contrast to DETR [5].

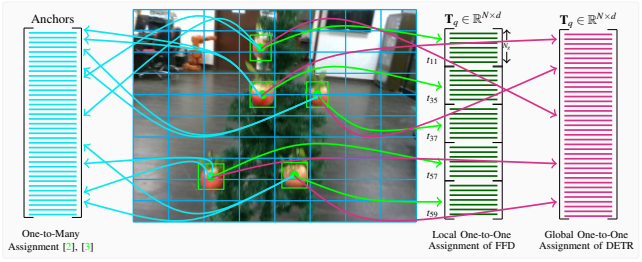

Fig. 4: FFD has novel query assignment. In traditional detectors [2], [3], a query is simply is an anchor. 't_{ij}' denotes a tile in the image.

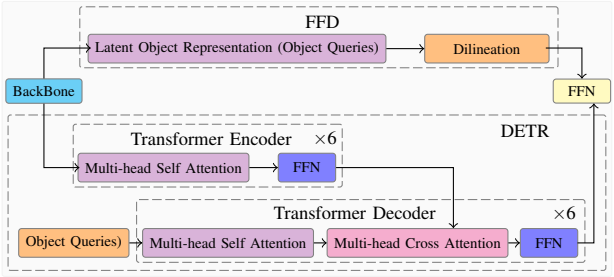

Fig. 5: Differences between FFD and DETR-like methods.

F. Tiled Hungarian Matching

Ground-truth matching is a crucial step to train an object detector which is performed via region proposal matching [2] and box mining [3]. It is full of hyperparameters and is a complicated process (Sec. II-A). To avoid that, we use bipartite matching using Hungarian algorithm inspired by [5] for assigning a ground-truth exactly one prediction, but performing it over tiles instead of the whole image space [5].

As mentioned previously that in our case, all of the N_g predictions for each tile are made w.r.t. the top-left corner of that corresponding tile, therefore to match a ground-truth box with a prediction, the prediction is denormalized via Eq. 7 and a cost is computed using L_{match} (discussed next). This is done for each ground-truth box whose center falls into that tile, resulting in a cost matrix $\mathscr{C} \in \mathbb{R}^{G \times N_g}$, where G denotes the number of ground truth boxes falling into a tile. Now, Hungarian matching is performed over $\mathscr{C}$ which assigns a ground-truth box exactly to one prediction $\in [0, N_g)$. This process is performed for all the tiles over the image.

$$\hat{b} = \{(b_{cx} - g_x)/g_w, (b_{cy} - g_y)/g_h, \log(b_w/W), \log(b_h/H)\} \quad (6)$$

$$b = \{\hat{b}_{cx} g_w + g_x, \quad \hat{b}_{cy} g_h + g_y, \quad \exp(\hat{b}_w) W, \quad \exp(\hat{b}_h) H\} \quad (7)$$

where, $\hat{b}$ is the prediction, b is denormalized box, W, H are the image width and height, and (g_x, g_y) is the top-left corner of the tile , and g_w, g_h are tile width and height respectively.

Tiled Hungarian matching is different from [5], [17]. *First*, since not all tiles are occupied, it prevents most tiles from performing the matching process, and *Second* not many objects are present in a tile. Together it drastically reduces matching complexity. The claims are verified in Table V.

G. Objective Function

The objective function is a weighted combination of a classification loss (Cross-Entropy) and a box regression (Smooth-$L1$) loss [2], formulated as below:

$$\mathscr{L}_c = -\log(p) \quad (8)$$

$$\mathscr{L}_b = \begin{cases} 0.5(b - \hat{p})^2/\beta, & \text{if}(b - \hat{b}) < 1 \\ (b - \hat{b}) - 0.5\beta, & \text{otherwise} \end{cases} \quad (9)$$

$$\mathscr{L} = \mathscr{L}_c + \lambda \mathscr{L}_b \quad (10)$$

where, λ is the loss weight which is set to 1, and balances the contribution of both losses. p and b are the class logits and box predictions, respectively. The overall objective $\mathscr{L}$ also serves as $\mathscr{L}_{match}$, which is used in the matching process.

H. Inference

Our inference strategy is free of any post-processing unlike popular approaches [2], [3], [4] due to the set predictions and one-to-one matching in contrast to the one-to-many assignment of [2], [3], [4] (discussed previously), resulting in the elimination of NMS entirely, and reduced CPU/GPU occupancy of FFD.

Further, in FFD, all the predictions are made w.r.t. the top-left corner of a tile, therefore they are denormalized by using Eq. 7 before the final use. The overall information flow of FFD is depicted in Fig. 3. Also, we have shown the difference between FFD and DETR [5] in Fig. 5.

IV. Occlusion Aware Scene Synthesis

CNN-based algorithms are sensitive to the amount of a dataset, if it is limited, the network may overfit and perform poorly. In our context, one image consists of several instances of apples which turns manual annotation of images an exhaustive and time-consuming task. Therefore, collecting many images and labelling them become a key challenge.

Hence we contribute by adapting occlusion-aware scene synthesis from our previous work [27]. The original approach generates realistic cluttered scenes from isolated object images when it is difficult to label real cluttered images. In this technique, an image called `base_image` is picked randomly from the dataset and is divided into a grid of $K \times K$. Now, another image from the dataset is chosen randomly, and pixels corresponding to an object instance in this image are pasted onto the grid center of one of the grids in the `base_image`. This procedure is repeated $K \times K$ grid locations. K is randomly chosen from 3×3, 4×4, 5×5 to simulate low, mid, and high clutter. Finally, instances below a visibility threshold (25%) are filtered out. See [27] for more details.

The data is then used for the task of semantic segmentation where it doesn't matter even if the object is visible by 25%. However, in object detection, as our objects are already too small, this approach generates cluttered images with too many overlapping and meaningless instances (Fig. 7a, 7b).

To adapt this approach to our use case, we make two changes. *First*, there is no notion of grids, instead maximum number of instances per image N_{max} is defined i.e. we use random locations instead of fixed grids, and *Second*, we put a constraint that none of the boxes overlap with each other.

In order to generate a synthetic scene based on the above changes, we begin by randomly selecting an image I_s^0 without any fruits (`base_image`). Then we randomly choose a

Fig. 6: (a) Indoor and outdoor tree dataset with and without fruit, (b) harvesting region dataset with and without fruit, and (c) synthetic scenes generated using the proposed occlusion aware scene synthesis.

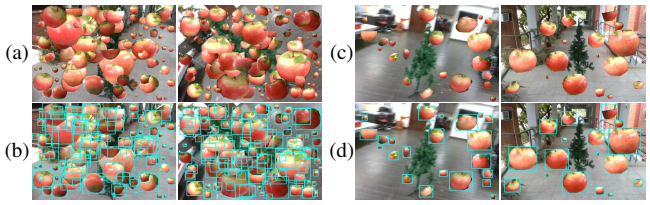

Fig. 7: (a) synthetic scenes generated by [27], (b) corresponding box annotations, (c) synthetic scenes generated by the improved approach, and (d) corresponding box annotations.

number $N_i \in [0, N_{max})$ which defines the number of instances the resulting synthesised image I_s will contain. Now, we randomly pick an image carrying fruit instances and its corresponding mask ground truth. With the help of the mask, the number of instances in this image is computed, and one of the instances is selected randomly to be transferred to the `base_image`. Now, a random location in I_s^0 is sampled, and before pasting the contents of the selected instance, it is ensured that the bounding box of this instance doesn't overlap with any of the instances already pasted during this process if placed at the sampled location. This procedure is repeated N_i times and can be summarized as:

$$\mathrm{I}_s^t = \mathrm{M}_{xy}^t * \mathrm{I}_s^{t-1} + \mathrm{P}_{xy}^t, \quad t \in (0, N_i] \tag{11}$$

$$s.t. \quad \mathrm{R}_{xy}^t \cap \mathrm{R}_{xy}^{t-1} = \phi \quad \forall\, t-1 \in (0, t) \tag{12}$$

where, $\mathrm{P}_{xy}^t, \mathrm{M}_{xy}^t, \mathrm{R}_{xy}^t$ refers to the patch, its mask, and its bounding box respectively. We set $N_{max} = 100$.

Fig. 7 shows synthetic scenes generated by the original [27] and our improved method. It can be noticed that the scenes generated by our method make much more sense in terms of visual quality as well as from the training perspective.

V. Dataset

Due to the lack of orchards in our vicinity, we build an farming setup (Fig. 1). It facilitates round-the-clock testing of aerial grasping without waiting for appropriate weather.

We collect two datasets: (`i`) D_t: fruit hanging over an artificial tree (Fig. 6a), and (`ii`) D_h: fruit hanging over the harvesting region (Fig. 6b). We collect 150 images for each case, both indoor and outdoor. Following [27], we also collect 25 images for each of the trees and the harvesting region without any fruit to serve as the `base_image` for generating synthetic scenes. In addition, we collect a few images of real trees with apples manually attached to it. It is done in order to test the robustness and generalization of FFD across scenes.

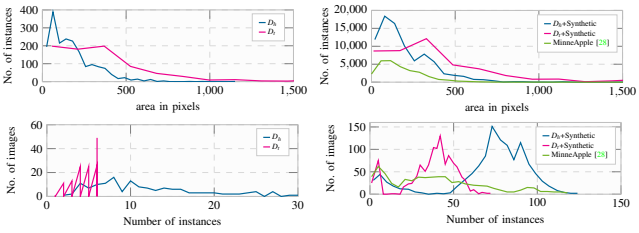

Fig. 8: Comparison of our dataset and MinneApple [28] benchmark.

Table I. Dataset statistics comparison.

Dataset	#Average Size (pixels)	#Average instance per image
D_h	13×13	13
D_t	20×20	6
D_h + Synthetic	14×14	**74**
D_t + Synthetic	20×20	40
MinneApple [28]	13×13	42

A. Labelling Process

For each image, a mask is generated whose pixels indicate class labels; Background has label 0 while fruit has label 1. Box annotations are extracted from the convex hull of the pixels belonging to an instance in the mask. Masks facilitate rotation augmentation since rotating a bounding box annotation does not precisely enclose the rotated object.

B. Synthetic Scenes

Manual labeling took 5-8 minutes per image of the harvesting region dataset due to a large number of instances, necessitating our scene synthesis technique (Sec. IV). Fig. 6c shows a few samples of synthesised scenes.

C. Comparison With Existing Benchmarks

Fig. 8 compares our dataset with the existing benchmark. The most closely related is the recent MinneApple [28] dataset consisting of images from apple orchards. It offers bounding boxes and masks for each instance. We see that, our dataset has many instances which are very small that are challenging for detectors. This is a unique aspect of our dataset.

Moreover, the MinneApple benchmark has two major issues. *First*, masks for many instances are missing, and *Second*, it also consists of several fruit instances which lie on the ground. The ground instances are not annotated, instead, only the ones on the tree are annotated. It results in an unfair evaluation because the ground instances resemble the ones on the tree and are detected by the detector. On the other hand, our dataset is free of such issues. See video.

Fig. 9: A few indoor and outdoor detection results. Boxes in red are the predictions while the ones in green are groundthruth.

Table II. Cross-dataset performance of FFD.

Exp	Train dataset	Test dataset	AP	AP_S	AP_M	AP_L
E1	D_t	D_t	51.1	28.6	51.7	65.7
		D_h	36.8	17.2	42.2	–
E2	D_h	D_t	30.2	23.5	34.6	19.1
		D_h	46.6	31.0	52.1	–
E3	D_t + D_h	D_t	53.9	23.1	55.2	67.4
		D_h	49.1	31.4	54.2	–

VI. Experiments

A. Training Hyperparameters

We set `base_lr` $= 0.001$, and use CosineAnnealing scheduler [29] with `weight_decay` $= 0.0001$, and ADAM optimizer with $\beta_1 = 0.90, \beta_2 = 0.99$ for 1000 epochs.

B. Comprehensive Data Augmentation

We use runtime augmentation [27] i.e. hue, saturation, brightness, and contrast perturbation with a likelihood of 0.4, random rotation in $[-10^o, 10^o]$, random translation in $[-50, 50]$ pixels, mirror, and scale. This prevents overfitting by accounting for lighting, and geometric transformations.

C. Training Policy

We split the datasets into a train-test ratio of $2:1$, while the outdoor images are used only for evaluation. We perform three experiments; *First*, E1: Train on D_t and test all, *Second*, E2: Train on D_h and test all, and *Third*, E3: Train on both D_t and D_h and test all. The resolution is set to 320×256.

D. Quantitative Evaluation

We report Average-Precision (AP) [2] to evaluate FFD, and AP_S, AP_M, AP_L for instances having different area (in pixels) i.e. small ($[0, 10^2]$), medium ($(10^2, 30^2]$), and large ($> 30^2$).

Table II shows the analysis of the three experiments. It can be seen that FFD performs with sufficiently high AP score, also verifiable via qualitative evaluations, discussed next.

It is interesting to note that cross-dataset testing has inferior performance when only one dataset is used for training (E1 or E2). As per our observations, FFD was able to detect all the instances on the cross-dataset, but AP dropped because of the misclassification of certain fruit-like spots in the images. It happened due to the lack of scenic diversity.

Furthermore, FFD has slightly higher AP in E1 relative to E2. This indicates the challenging nature of harvesting region dataset due to the presence of small instances.

E3 shows that using both datasets improves the accuracy for each of them, owing to the increased image diversity.

Table III. Detection Performance on our dataset. FFD has very high detection performance while still being faster than all the baselines.

Detector	AP	AP_S	AP_M
SSD @multi-scale [3]	38.0	20.1	39.1
DETR @multi-scale [5]	40.2	24.9	43.0
FCOS @multi-scale [4]	42.1	26.8	46.9
Faster-RCNN @multi-scale [2]	45.9	28.7	51.5
YOLO-v8 @multi-scale [30]	46.3	29.2	51.5
YOLO-v8 @single-scale [30]	35.2	23.1	41.4
FFD @single-scale	**46.6**	**31.2**	**52.1**

1) Qualitative Results: Fig. 9 shows a few detection samples from the test-set, and also detections on outdoor images which none of the experiments used for training. From the detection quality, it can readily be verified that the detections have a very high overlap with the ground-truth boxes, which is a most required attribute for robotic harvesting autonomy. This facilitates accurate centroid calculation using depth information, a crucial step for performing a robust visual servoing and grasping operation using UAV.

E. Detection Performance Against Exiting Detectors

We compare FFD with popular and well-established detectors by customizing them for our dataset. This task itself is challenging because each detector has its source code implemented differently in different frameworks. This raises the difficulty level to analyze each of them. Hence the baselines are selected such that it covers almost all the varieties of the detectors, i.e. multi-stage [2], single stage [3], [4] and transformer-based [5] to minimize the retraining efforts. We leave DETR's successor [17] due to its highly complex and resource-hungry training, and [11] due to its backbone-only modifications and traditional detection process.

1) Detection on Our Dataset: Table III shows the corresponding analysis. The baselines are trained with our backbone (Sec. III-A), on the harvesting region dataset due to its higher difficulty. From the table, we can see that FFD is as accurate as the most complex detector Faster-RCNN [2] and the latest YOLO-v8, including small objects. But FFD outperforms them in single-scale comparison, i.e. when Faster-RCNN and YOLO are trained for single-scale detection only similar to FFD. It even performs better than the transformer-based DETR [5], as DETR converges slowly; however, it can be trained longer to achieve comparable accuracy.

FFD earns this upper hand only because of the LOR module, precisely due to the query generation from the feature map and the prediction strategy, which is the main novelty of FFD, along with its unique training scheme.

Table IV. Evaluation on MinneApple [28]. '*' denotes our results. All networks are trained with ResNet-50 backbone with 25M parameters. Our backbone variant of FFD has only 3M parameters.

Method	Backbone (#Params)	AP	AP_S	AP_M	AP_L
Tile-Faster-RCNN [28] @multi-scale	ResNet-50 (25M)	34.1	19.7	51.9	20.8
Faster-RCNN [2] @multi-scale	ResNet-50 (25M)	42.3	27.9	58.5	88.2
YOLO-v8 @multi-scale [30]	CSPDarkNet (25M)	44.3	28.3	60.3	84.3
Faster-RCNN [2] @single-scale	ResNet-50 (25M)	33.9	15.2	51.2	61.9
YOLO-v8 @single-scale [30]	CSPDarkNet (25M)	36.6	18.9	54.7	65.6
DETR [5]* @single-scale	ResNet-50 (25M)	15.2	8.1	19.8	10.6
FFD @32 × 32 @single-scale	ResNet-50 (25M)	**44.6**	**29.5**	**60.5**	**92.2**
FFD @32 × 32 @single-scale	our backbone (3M)	30.1	21.8	48.8	60.5

Table V. Training and inference runtime at full precision (FP32). FFD-C++ denotes "C++" implementation. Training is done on NVIDIA RTX-2070, and inference on NVIDIA Jetson Xavier NX.

Model	Faster-RCNN [2]	SSD [3]	FCOS [4]	DETR [5]	YOLO-v8 [30]	FFD	FFD-C++
Per iteration Training Time	5.10s	4.60s	3.40s	6.80s	0.95s	**0.70**s	**0.40**s
Inference @FP32	49ms	32ms	30ms	25ms	29ms	**20**ms	**11**ms

Note: Accuracy can be improved by changing the backbone. We fixed the backbone and kept sufficiently large epochs to meet our speed and resource requirements.

2) Detection on MinneApple Benchmark [28]: We also conduct experiments on the recent MinneApple benchmark for apple detection (See Table IV). Noticeably, with ResNet-50 backbone and different tile sizes, FFD achieves similar detection scores at single-scale detection while being considerably faster. Most importantly, Faster-RCNN performs multi-scale detection [28], which is still slower than FFD. Moreover, the latest YOLO-v8 performs worse in single-scale settings but is comparable to FFD in multi-scale. This shows the uniqueness of FFD that despite being single-scale, it outperforms multi-scale methods.

Since MinneApple is a challenging dataset, it needs a bigger backbone. However, we also tried our smaller backbone, which obtains a lower AP. It is evident due to its fewer parameters, i.e. only 3M vs 25M of ResNet-50.

F. Fastest Training

Table V shows the training efficiency on an NVIDIA RTX-2070 GPU. Interestingly, FFD has the lowest per-loop training time, primarily attributed to our proposed query assignment and matching strategy, and performing fewer matches relative to the large number of matches in multi-scale detection [2].

G. Runtime Efficiency Gains

We report runtime analysis over NVIDIA Jetson Xavier NX, a 10W palm-sized embedded computing device with 384 CUDA cores @FP32 precision.

1) Faster Resource Exemption: Table V shows the runtime analysis of different methods with our backbone. It can be seen that FFD is the fastest among all the algorithms. The primary reasons are its minimal architectural components and no multi-scale detection, making FFD a simpler and post-processing-free pipeline.

Runtime is a key metric which determines the duration for which GPU resources shall be held by the detector. From the table, it can be readily seen that FFD has the minimum hold time i.e. 11ms which is significantly lower than the baselines and is a major achievement and motivation of this work.

Table VI. Effect of synthetic scenes (S.S.) & augmentation.

Colour	Scale	Mirror	Rotate	S.S.	AP	AP_S	AP_M
✓	✓				0.08	0.05	0.2
✓	✓	✓			8.1	0.9	14.7
✓	✓	✓	✓		13.6	6.3	15.6
✓	✓	✓	✓	✓(Ours)	46.6	31.0	52.1
✗	✗	✗	✗	✓(Ours)	45.1	34.8	49.2
✗	✗	✗	✗	✓ [27]	30.7	31.7	63.9

2) Resource Allocation to Co-Existing Sub-Systems: It should be noticed that FFD has a very high speed, but during deployment, images from the sensor/camera can be obtained only at a rate of 30Hz. However, the high speed ensures the consumption of computing resources for a small duration so that the other compute-intensive algorithms can utilize them. For this reason, even when FFD and other compute-intensive tasks are concurrently running, it does not affect the desired FPS because a lot of computational space is still left on the device. On the other hand, in the existing methods, if two algorithms are deployed simultaneously, each of the algorithms affects the speed of the others because resources are being used for too long. Hence, achieving higher speeds is necessary to guarantee freeing computing resources in a timely manner. Eliminating the post-processing step also reduces the power consumption and programming complexity in contrast to the standard pipelines, which is an additional crucial objective for deployment.

H. Ablation Study

1) Synthetic Scenes & Comprehensive Data Augmentation: Table VI shows the effect of proposed occlusion-aware scene synthesis along with comprehensive data augmentation on the harvesting region train-test split (D_h).

Noticeably, synthetic scenes alone help achieve high accuracy, while using them with data augmentation further improves the performance. Without augmentation, FFD exhibits overfitting, which is intuitive because of the small dataset. Our findings are consistent with [1], which mentions the benefits of employing these techniques in the training.

We also compare our scene synthesis technique with the original one [27] (Table VI). Noticeably, AP decreases for [27], which is in accordance with our claim in Sec. IV.

2) Tile Size: The number of queries is determined by the number of predictions per tile (N_g) and tile-size. Hence, it is important to see an ablation of how the performance of FFD varies with this parameter. We provide this analysis in Table VII by varying the tile-size which is selected such that image resolution can be divided with zero remainder.

We accommodate different tile-sizes by changing the strides in the final stage. For 16×16 tile-size, the final stage operates at a unit stride, resulting in $H_o = \frac{H}{16}$, $W_o = \frac{W_o}{16}$, while in 64×64, the last two layers of the final stage operate at a stride 2, resulting in $H_o = \frac{H}{64}$, $W_o = \frac{W_o}{64}$,

From the experiment, we analyzed that as feature resolution is reduced, accuracy decreases. Accuracy remains stable up to 32×32 tile-size, and then decreases significantly.

Table VII. Effect of tile-size.

$S(\cdot)$	#Params	N_q	Runtime (ms)	AP	AP_S	AP_M
16×16	3.1M	1600	14ms	45.3	32.7	50.0
32×32	3.4M	800	11ms	46.6	31.0	52.1
64×64	6.5M	400	13ms	10.3	3.1	12.4

Table VIII. Effect of squeezing type $S(\cdot)$ in CCGC.

$S(\cdot)$	AP	AP_S	AP_M
Sigmoid	46.6	31.0	52.1
Softmax	10.0	0.1	12.5

While keeping the tile-size to a very small number increases the computations in the backbone for the same number of parameters and more queries. Hence, based on the runtime goals, tile-size can be kept to 32×32 regardless of the resolution depending upon the requirements.

3) Effect of Squeezing Type in CCGC: CCGC is a crucial component of the LOR module and uses sigmoid by default. However, it is important to analyze the effect of different squeezing activation. We conduct this experiment by replacing sigmoidal activation with softmax operation.

We observe that softmax faces convergence issues in the same training time (see Table VIII). In addition, from a speed perspective, sigmoid is always faster than softmax since it does not require the normalization step.

VII. Conclusion

This work introduces a Fast-Fruit-Detector (FFD) for UAV based fruit harvesting task in vertical farming setting. The paper mainly focuses on the visual perceptions system. A deep learning based single-stage, post-processing free object detector "FFD" has been proposed which can run at 100FPS on Jetson Xavier NX @FP32 precision and above 200FPS @FP16 or Int8. FFD neither requires multi-scale feature fusion to detect small object nor requires post-processing such as NMS which is accomplished via novel components of FFD; latent object representation module (LOR), and query assignment and prediction strategy. In addition, we present an approach to generate synthetic scenes to avoid exhaustive manual effort for labeling of fruit images. We thoroughly asses FFD on a variety of indoor-outdoor scenes which suggests that FFD outperforms various mainstream detectors in terms of training-testing efficiency, and accuracy evaluation. FFD is not limited only to this purpose, but can be adapted to other robotic applications as well.

References

[1] A. Kumar, M. Vohra, R. Prakash, and L. Behera, "Towards deep learning assisted autonomous uavs for manipulation tasks in gps-denied environments," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1613–1620, IEEE, 2020.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[4] Z. Tian, C. Shen, H. Chen, and T. He, "FCOS: Fully convolutional one-stage object detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9627–9636, 2019.

[5] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *ECCV*, pp. 213–229, Springer, 2020.

[6] C. Zhu, Y. Zheng, K. Luu, and M. Savvides, "CMS-RCNN: contextual multi-scale region-based cnn for unconstrained face detection," in *Deep Learning for Biometrics*, pp. 57–79, Springer, 2017.

[7] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[8] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," *CVPR*, 2017.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[10] Y.-M. Zhang, C.-C. Lee, J.-W. Hsieh, and K.-C. Fan, "CSL-YOLO: A new lightweight object detection system for edge computing," *arXiv preprint arXiv:2107.04829*, 2021.

[11] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLO-v4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[12] Y. Li, J. Li, W. Lin, and J. Li, "Tiny-DSOD: Lightweight object detection for resource-restricted usages," *arXiv:1807.11013*, 2018.

[13] P. Roy and V. Isler, "Surveying apple orchards with a monocular vision system," in *2016 IEEE international conference on automation science and engineering (CASE)*, pp. 916–921, IEEE, 2016.

[14] I. Sa, Z. Ge, F. Dayoub, B. Upcroft, T. Perez, and C. McCool, "Deepfruits: A fruit detection system using deep neural networks," *sensors*, vol. 16, no. 8, p. 1222, 2016.

[15] S. Bargoti and J. Underwood, "Deep fruit detection in orchards," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3626–3633, IEEE, 2017.

[16] P. Roy, A. Kislay, P. A. Plonski, J. Luby, and V. Isler, "Vision-based preharvest yield mapping for apple orchards," *Computers and Electronics in Agriculture*, vol. 164, p. 104897, 2019.

[17] S. Liu, F. Li, H. Zhang, X. Yang, X. Qi, H. Su, J. Zhu, and L. Zhang, "DAB-DETR: Dynamic anchor boxes are better queries for DETR," *arXiv preprint arXiv:2201.12329*, 2022.

[18] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[20] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, pp. 448–456, 2015.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[22] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[23] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, 2019.

[25] F. Li, H. Zhang, S. Liu, J. Guo, L. M. Ni, and L. Zhang, "DN-DETR: Accelerate DETR training by introducing query denoising," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13619–13627, 2022.

[26] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[27] A. Kumar and L. Behera, "Semi supervised deep quick instance detection and segmentation," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8325–8331, IEEE, 2019.

[28] N. Häni, P. Roy, and V. Isler, "Minneapple: a benchmark dataset for apple detection and segmentation," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 852–858, 2020.

[29] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[30] "YOLO-v8," in *https://github.com/ultralytics/ultralytics*.