

OBJECT DETECTION AND PERSON TRACKING USING UAV

Submitted in partial fulfillment of the requirements for the award
of Bachelor of Engineering Degree in Computer Science and Engineering

By

Manjunatha inti

(38110301)

Manojpawar S J



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade "A" by NAAC
JEPPIAR NAGAR, RAJIV GANDHI SALAI, CHENNAI-600119**

MAY 2022



SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Professional Training Report is the Bonafede work of **Manjunatha Inti (38110301)**, **Manojpawar SJ (38110306)** who carried out the project entitled "**OBJECT DETECTION AND PERSON TRACKING USING UAV**" under our supervision from SEPTEMBER 2021 to MARCH 2022.

**Internal Guide
Dr. A. Christy Ph.D.**

Head of the Department

Submitted for Viva voce Examination held on_____

Internal Examiner

External Examiner

DECLARATION

I Manjunatha Inti (38110301), Manojpawar SJ (38110306) hereby declare that the Project Report entitled "OBJECT DETECTION AND PERSON TRACKING USING UAV" done by us under the guidance of **Dr. A. Christy Ph.D.** at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE: 30-03-2022

SIGNATURE OF THE CANDIDATE

PLACE: CHENNAI

MANJUNATHA INTI
MANOJPAWAR SJ

ACKNOWLEDGEMENT

I am pleased to acknowledge our sincere thanks to **Board of management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey our thanks to **Dr. T. Sasikala, M.E., Ph.D.**, Dean, School of Computing, **Dr. S. VIGNESHWARI, M.E., Ph. D.**, **Dr.L.Lakshmanan M.E.,Ph.D.**, Heads of the Department, for providing me necessary support and details at the right time during the progressive reviews.

We would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. A. Christy Ph.D.**, for her valuable guidance, suggestions, and constant encouragement paved way for the successful completion of my project work.

We wish to express our thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Today Object Detection and classifying objects inside a single frame that contains many objects is a time-consuming task. The accuracy rate has grown dramatically as a result of the deep learning technique. Despite new flight control laws, Unmanned Aerial Vehicles (UAVs) continue to grow in popularity for civilian and military uses, as well as personal use. This growing interest has accelerated the development of effective collision avoidance technologies. Such technologies are crucial for UAV operation, particularly in congested skies. Due to the cost and weight constraints of UAV payloads, camera-based solutions have become the de facto standard for collision avoidance navigation systems. This requires multitarget detection techniques from video that can be effectively run on board.

A drone is a quadcopter with on board sensors. This drone can be controlled using wi-fi and laptop, using a programming language called Python and a Python library called as drone kit. This paper will discuss a way for tracking a specific object known as object detection tracking methods, which may be used to track any arbitrary object chosen by the user, the camera of drone is used to take video frames along with all the sensor's information such as ultrasonic sensors, GPS etc. A train model will identify an object first and determines the direction at which the drone should fly so that it keeps following person.

KEYWORDS - UAV, drone, Single shot detector, Mobile-Net, OpenCV, etc.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	5
	LIST OF FIGURES	8
1	INTRODUCTION	
	1.1 OUTLINE OF THE PROJECT	9
	1.2 LITERATURE REVIEW	10
	1.3 OBJECTIVE OF THE PROJECT	12
2	AIM AND SCOPE OF THE PRESENT INVESTIGATION	
	2.1 AIM AND SCOPE OF THE PROJECT	13
	2.2 HARDWARE REQUIREMENTS	13
	2.3 SOFTWARE REQUIREMENTS	13
3	EXPERIMENTAL OR MATERIALS AND METHODS, ALGORITHMS USED	
	3.1 SYSTEM DESIGN	15
	3.1.1 EXISTING SYSTEM	16
	3.1.2 PROBLEM STATEMENT	16
	3.1.3 PROPOSED SYSTEM	16
	3.2 METHODS USED	17
	3.2.1 TAKE OFF	17
	3.2.2 SEARCHING FOR THE OBJECT TO TRACK	18
	3.2.3 FIND THE OBJECT AND PRINT ITS BOUNDING	18
	3.2.4 ADJUST YAW, VELOCITY AND ANGLE TO TRACK	
	3.2.5 LAND	18
	3.3 ALGORITHMS	
	3.3.1 Mobilenet SSD	
	3.3.2 Result	

4	RESULTS AND DISCUSSION	
	4.1 Results	26
5	CONCLUSIONS	28
6	REFERENCES	29
7	APPENDIX	
		38
	7.1SOURCE CODE	32
	7.2SCREENSHOTS	
	7.3PUBLICATION WITH PLAGIARISM REPORT	47

LIST OF FIGURES

FIG NO	TITLE	PAGE NO
1.1	INTRODUCTION	9
1.3	A typical UAV analysis model	12
3.1	The overall procedure diagram of the object detection and person tracking	15
3.3.1	Mobilenet SSD	18
3.3.1	Overview of models	21
3.3.2	Module Result	24
7.2	Screen shots	45

1. INTRODUCTION

1.1 OUTLINE OF THE PROJECT:

Recently Drones are becoming affordable, practical, and versatile, which is resulting in a reality where Drones are widely available in the sky for commercial and individual purposes. The UAV's has become more advantageous in the compact structure and low maintains and increased processing ability. In order to accomplish robust tracking on drones, UAVs outfitted with high-resolution cameras and running computer vision algorithms in real-time have recently received considerable study focus Annotations of high quality on large datasets are essential for the development of algorithms, therefore specific difficulties related to object detection are addressed. and person tracking algorithms applied to UAV applications involve the following. The video feed is acquired from a moving camera mounted on the UAV, where we detect the person in the drone camera and capture his activities for surveillance purpose, it can be used in logistics in remote location as well as cinematography etc. To achieve this task, we'll use the AR Drone's built-in sensors and actuators, but due to the built-in processor's limited computing power, we'll do our reasoning on an external computer (a regular laptop). The usage of Image Classification with the Mobile Net model to accomplish detection and classification is the subject of this paper. The entire system works by taking a frame from the camera and then using the on-board computer to detect an object. computer processes the image to detect the object using a Mobile-net classifier. Once the object is detected, the on-board goal is to predict the object's position in relation to the quadrotor and transmits the appropriate information to the flight controller to ensure proper tracking.

1.2 LITERATURE REVIEW

[1] Multi-Inertial Sensing Data for Real-Time Object Detection and tracking on a Drone To extract features from an image, this study employs the Oriented Fast and Rotated BRIEF[ORB] algorithm, as well as the Euclidean equation GPS and IMU to compute the relative position between the drone and the target.

[2] Target Tracking and Recognition Systems Using Unmanned Aerial Vehicles. This paper uses YOLO algorithm with custom dataset and train the algorithm for motion blurred images and low resolution.

[3] Multi-Target Detection and Tracking in Unmanned Aerial Vehicles with a Single Camera (UAVs). The Lucas-Kanade method is used in this research to recognize and track other fast-moving UAVs.

[4] Object Detection and Classification for Autonomous Drones. This paper aims to implement object detection and classification with high accuracy using SSD architecture combined with Mobile Net

[5] Agent Sharing Network with Multi-Drone based Single Object Tracking. This paper uses Agent Sharing Network [ASNet] for multiple drones to track and identify single object.

[6] Path Following with Quad Rotorcraft Switching Control: An Application This paper focus on estimation of track and road using UAV and here they used visual sensors to identify lane.

[7] Any flying drone can track and follow any object. The Drones may track an arbitrary target selected by the user in the video stream coming from the drone's front camera. The proportional-integral-derivative method is then used to direct the drone based on the location of the monitored object (PID controller). They employed a tracking-learning-detection technique in computer vision.

[8] Haar-like Features for Object Recognition and Tracking Application of Cascade classifiers on a quad-rotor UAV. In this research, to develop a functioning Unmanned Aerial Vehicle (UAV) capable of tracking an object, we used a Machine Learning-like vision system called Haar-similar features classifier. On-board image processing is handled by a single-board computer with a powerful processor.

[9] Object Detection in Real-Time (YOLO) approach, which is based on UAV, has been retrained to swiftly and accurately detect and distinguish objects in UAV photos.

[10] Drone Identification and Tracking Using Phase-Interferometric Doppler Radar This work focuses on UAV detection and tracking using phase interferometric doppler radar, where data is collected using a double channel doppler and multi radar called Doppler-azimuth, which will handle the processing.

[11] Deep Learning-Based Object Detection for Quadcopter Drones. The evolution of object detection in AI and deep learning-based drone cameras is discussed in this study. the main aim of this paper is delivering medical aids to patients. Here they used single shot detector and Mobile Net.

[12] With an embedded UAV, real-time visual object detection and tracking is possible. This is true even for embedded devices. A powerful neural network-based object tracking system could be deployed in real time. A modular implementation suited for on-the-fly execution is described and evaluated (based on the well-known Robot Operating System) SYSTEM ARCHITECTURE.

[13]. Deep Learning-Based Object Detection for Quadcopter Drones Widodo Budiharto¹, Alexander A S Gunawan², Jarot S. Suroso³, Andry Chowanda¹, Aurello Patrik¹, and Gaudi Utama¹ ¹Computer Science Department, Bina Nusantara University's School of Computer Science.

[14]. Convolutional Neural Networks with Time Domain Motion Features for Drone Video Object Detection Yugui Zhang¹ Liuqing Shen¹ Xiaoyan Wang¹ Hai-Miao Hu¹, ² ¹ Beijing Key Laboratory of Digital Media, School of Computer Science and Engineering ² State Key Laboratory of Virtual Reality Technology and Systems

Beihang University, Beijing, China

1.3 OBJECTIVE OF THE PROJECT

This study was conducted based on several objectives which are:

- a) To record and detect human activities for surveillance purpose,
- b) It can be used in logistics in remote locations, and used in cinematography and photography etc.
- c) Through using general unmanned aerial vehicle or drone and computer vision.

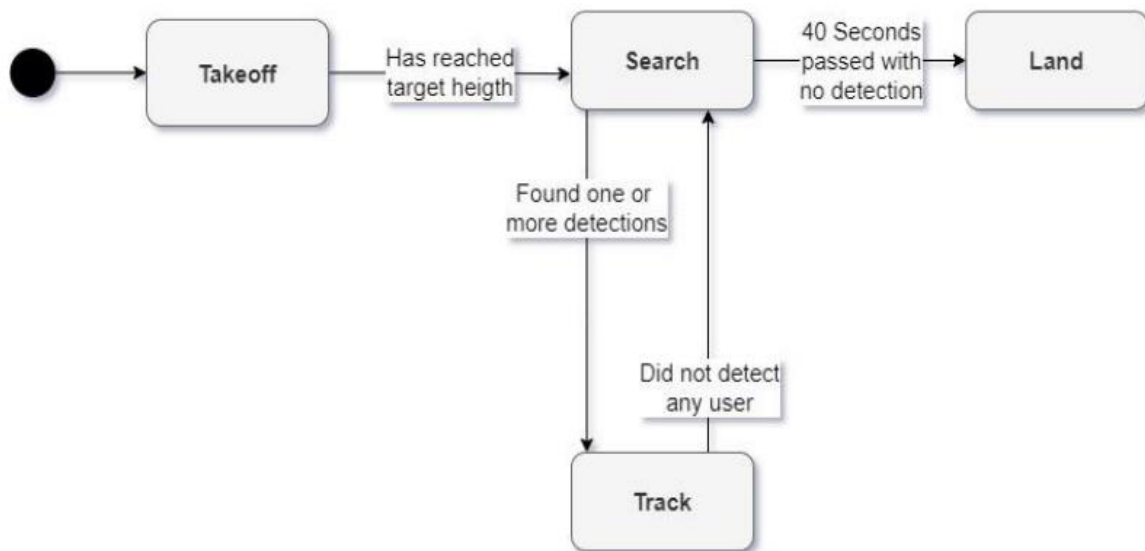


Figure 1. A typical UAV analysis model

2.AIM AND SCOPE OF THE PRESENT INVESTIGATION

2.1 AIM AND SCOPE OF THE PROJECT

The Aim of the project is based on object detection and person tracking. In this process we get the real time input through OpenCV. video is converted into images then the image is processed in open cv method. When the object detection takes place then the model will be able to identify human from objects, if human is in the video then UAV keeps on following for certain limit.

The scope of the project is working with a large number of features as it can capture video frames from long range that's the biggest advantage that can be used in surveillance purpose. Even, it has also the risk of overfly or any failure for that purpose we involved kill switch it going to terminate UAV, we used raspberry pi and APM 2.8 flight controller to perform all the necessary tasks.

2.2 HARDWARE REQUIREMENTS

- UAV: The main elements of our quadcopter includes Battery, APM 2.8 flight controller, GPS module, Raspberry-pi on board computer, USB camera, Ultra sonic sensor.
- APM 2.8 flight controller: it has ATMEGA 2560 processor and having 3 axis gyro meters on board, accelerometer and barometer by using this sensor on circuit board we can use this to control any UAV's.
- We use external GPS for pin point accuracy to find coordinates.
- Raspberry-pi is on board computer which run our python scripts, get frames from USB camera and send signals to flight controller.
- Ultra-sonic sensor: This sensor uses ultra-sonic sound to get distance.

2.3 SOFTWARE REQUIREMENTS

- Host computer: The host computer will reserve all the input frames from camera on raspberry pi and process each frame using OpenCV python for object detection and classification. Once the object is detected the Mobile Net classifier will determine the bounding boxes around the object and thus we can identify those objects along with labels.
- Mobile Net: There are 2 forms of blocks in MobileNetV2. One could be a one-stride residual block. Another choice for shrinking could be a block with a two stride. Both forms of blocks have 3 levels. Eleven convolution victimization is the first layer. These points should be reLU6ed. The second layer has a depth-wise convolution. Another eleven convolutions are employed inside the 3rd layer, However, there is no non-linearity at this time. Deep networks can only have the capability of a learning algorithm on the non-zero region as a

component of the output region if RELU is applied again, according to the statement.

- Drone Kit: Drone kit is a python library. Developers can use drone Kit-Python to make programmed that operate using an onboard companion computer, and use a low-latency link to communicate with the Arduino control board. Onboard apps can assist the autopilot in performing computationally hard or time-sensitive tasks, as well as contributing knowledge to the vehicle's action. Drone Kit python can also be used by base station apps that interface with vehicles over a higher latency RF-link.
- MAV-Link is used by the API to communicate with automobiles. By offering application support to a connected vehicle's data, status, and parameter information, it offers either task management or total control on vehicle movement and operations.

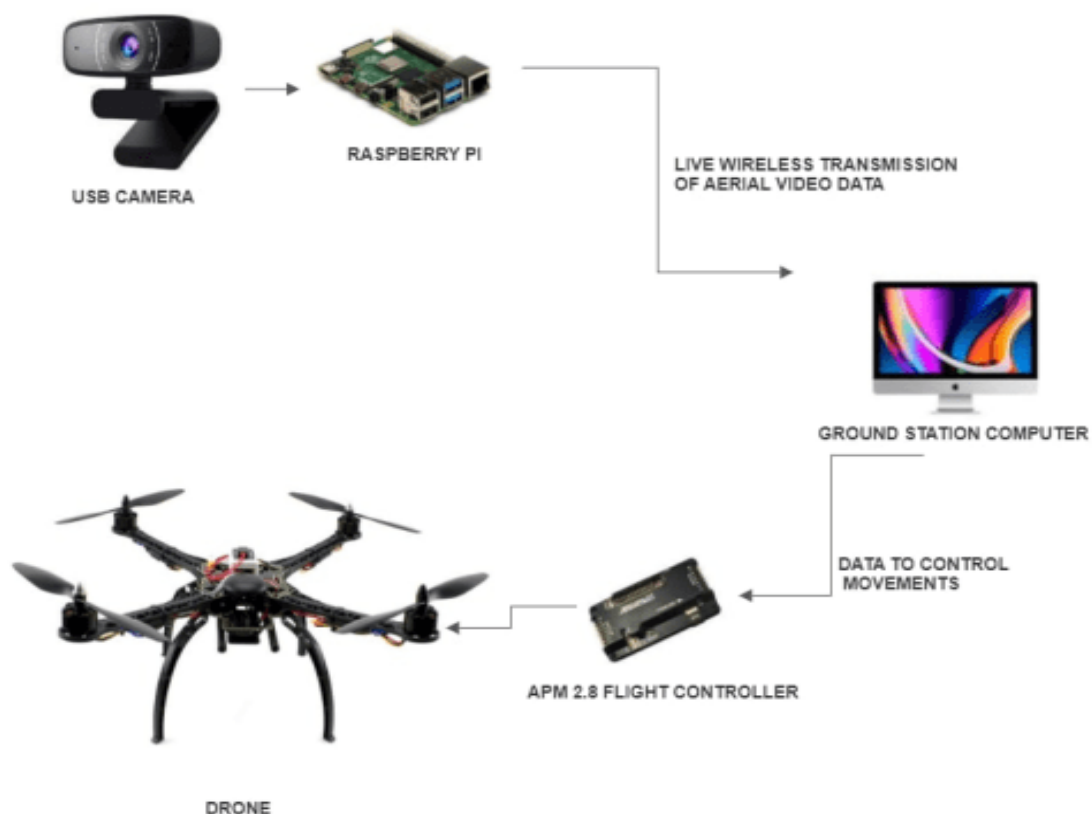


Figure .2 System Architecture

3. EXPERIMENTAL OR MATERIALS METHODS, ALGORITHMS USED

3.1 SYSTEM DESIGN

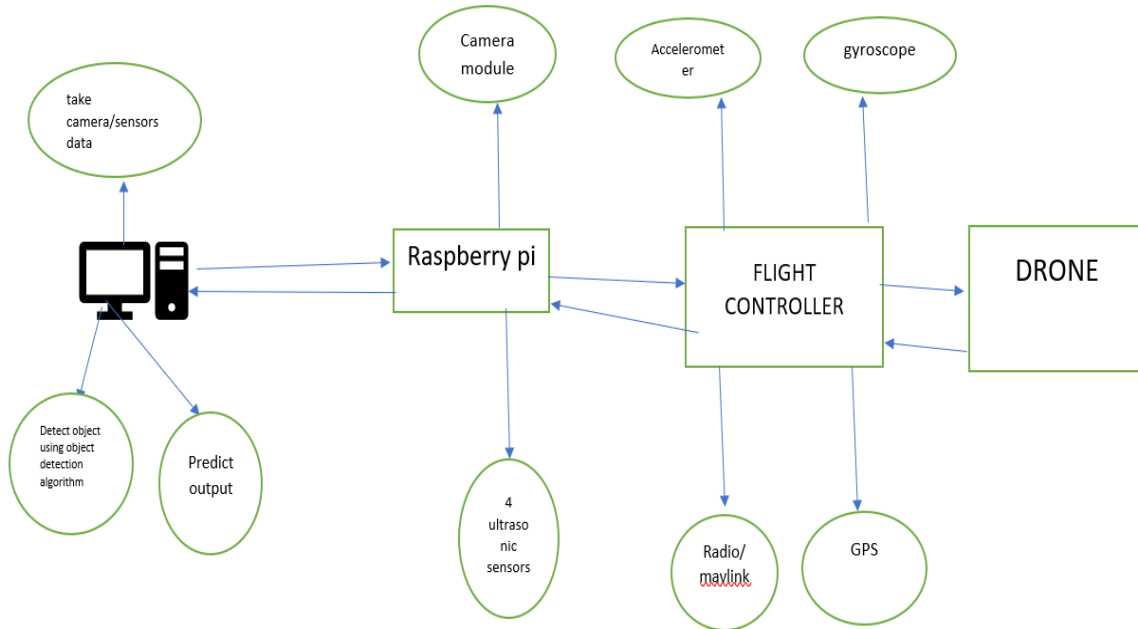


Figure 2. The overall procedure diagram of the drone object detection

This allows us to immediately discover and classify accidents in order to dispatch assistance and save those who are injured.

The need for object detection systems is increasing due to the ever-growing number of digital images in both public and private collections. Object recognition systems are important for reaching higher-level autonomy for robots [3]. Applying computer vision (CV) and machine learning (ML), it is a hot area of research in robotics. Drones are being used more and more as robotic platforms. The research in this article is to determine how to use existing object detection systems and models can be used on image data from a drone. One of the advantages of using a drone to detect objects in ascend may be that the drone can move close to objects compared to other robots, for example, a wheeled robot. However, there are difficulties with UAVs because of top-down view angles and the issue to combine with a deep learning system for compute-intensive operations. When a drone navigates a scene in search for objects, it is of interest for the drone to be able to view as much of its surroundings as possible. However, images taken by

UAVs or drones are quite different from images taken by using a normal camera. For that reason, it cannot be assumed that object detection algorithms normally used on “normal” images perform well on taken by drone images. Previous works on this stress that the images captured by a drone often are different from those available for training, which are often taken by a hand-held camera. Difficulties in detecting objects in data from a drone may arise due to the positioning of the camera compared to images taken by a human, depending on what type of images the network is trained on. In the previous research, the aim of work was to show whether a network trained on normal camera images could be used on images taken by a drone with satisfactory results. They have used a fish-eye camera and conducted several experiments on three kinds of datasets such as images from a normal camera, images from a fish-eye camera, and rectified images from a fish-eye camera

3.1.1 EXISTING SYSTEM:

- There are many solutions proposed for the concerned problem and each one has some advantage over others. Among them are traditional machine learning algorithms for finding and identifying objects
- Our system uses more than one sensor to increase the accuracy of the system and we are also using Mobilenet SSD detector for good accuracy on low end companion devices like raspberry pi to cut down cost
- Our system cuts the unnecessary cost by using the already existing infrastructure like camera, accelerometer, barometer etc. sensors.

3.1.2 PROBLEM STATEMENT:

- Not found any cost-effective solutions in this research.
- Mobilenet SSD detection algorithm is used to identify the nearby objects upto 100 classes on MSCOCO dataset
- fast process to detect objects.
- average accuracy in this project.

3.1.3 PROPOSED SYSTEM:

- This project is based on object detection and tracking using UAV.

- In this process we get the input as video.
- Video is converted to image by using open cv method.
- Then the image is processed in open cv method.
- When the object is detected print the object name in the output.
- Then adjust height and speed of the UAV for tracking the object.

ADVANTAGE:

- Working with a large number of features may affect the performance because training time increases exponentially with the number of features.
- Even, it has also the risk of over fitting with the increasing number
- Getting a more accurate prediction, feature selection is a critical factor here.

3.2 METHODS USED:

- Take OFF
- Searching for the objects to track
- Find the object and print its bounding box
- Adjust yaw, velocity and angle to track movements of that object
- Land

3.2.1 TAKE OFF:

- Initially UAV will be armed by using a remote.
- UAV will go through all the necessary checks on sensors in order to ensure that all the sensors are working.
- If checks are failed then drone will be shut down immediately with an error message
- If all the checks are passed then we have set the drone to fly 1.5 meters above ground holding the constant altitude.

3.2.2 SEARCHING FOR THE OBJECT TO TRACK:

- We will start our python script for object detection.
- The UAV will start rotating 360 degree slowly so that it can find objects to track.
- If UAV finds an object to track then it will auto adjust its height and velocity.

3.2.3 FIND THE OBJECT AND PRINT ITS BOUNDING BOX:

- When our python script detects an object will will print the bounding box around the object.
- It will also print the class name or the name of the object
- Under the hood we are using Mobilenet SSD detector to finding and calculating bounding boxes and class names of the object

3.2.4 ADJUST YAW, VELOCITY AND ANGLE TO TRACK:

- IF our mobilenet algorithm finds and object we will obtain its x, y, z coordinates in 2-Dimensional frame.
- We will subtract the x, y, z coordinates from the center of the frame
- Depending on the sign and distance we will send Command to UAV to move on that direction and distance

3.2.5 LAND:

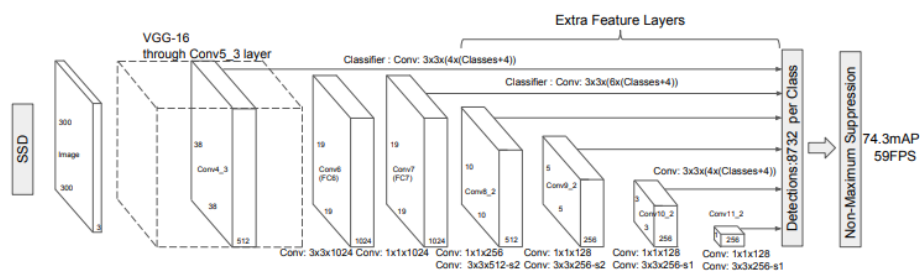
- If our mobilenet detection algorithm did not find any object in the 40 sec time then we will trigger a Return to Launch command to UAV so that it can land Safely on the launchpad

3.3 ALGORITHMS USED:

3.3.1 Mobilenet SSD:

MobilenetSSD is an object detection model that computes the bounding box and category of an object from an input image. This Single Shot Detector (SSD) object detection model uses Mobilenet as backbone and can achieve fast object

detection optimized for mobile devices.



MobilenetSSD takes a (3,300,300) image as input and outputs (1,3000,4) boxes and (1,3000,21) scores. Boxes contains offset values (cx,cy,w,h) from the default box. Scores contains confidence values for the presence of each of the 20 object categories, the value 0 being reserved for the background. In SSD, after extracting the features using an arbitrary backbone, the bounding boxes are calculated at each resolution while reducing the resolution with Extra Feature Layers. MobilenetSSD will concatenate the output of the six levels of resolution and calculate a total of 3000 bounding boxes, and finally, filter out bounding boxes using non-maximum suppression (nms).

By using SSD, we only need to take one single shot to detect multiple objects within the image, while regional proposal network (RPN) based approaches such as R-CNN series that need two shots, one for generating region proposals, one for detecting the object of each proposal. Thus, SSD is much faster compared with two-shot RPN-based approaches.

SSD is designed for object detection in real-time. Faster R-CNN uses a region proposal network to create boundary boxes and utilizes those boxes to classify objects. While it is considered the start-of-the-art in accuracy, the whole process runs at 7 frames per second. Far below what real-time processing needs. SSD speeds up the process by eliminating the need for the region proposal network. To recover the drop in accuracy, SSD applies a few improvements including multi-scale features and default boxes. These improvements allow SSD to match the Faster R-CNN's accuracy using lower resolution images, which further pushes the speed higher. According to the following comparison, it achieves the real-time processing speed and even beats the accuracy of the Faster R-CNN.

(Accuracy is measured as the mean average precision mAP: the precision of the predictions.)

We have implemented Mobilenet SSD using pretrained weights using opencv.

- **Multi-scale feature maps for detection**

At first, we describe how SSD detects objects from a single layer. Actually, it uses multiple layers (multi-scale feature maps) to detect objects independently. As CNN reduces the spatial dimension gradually, the resolution of the feature maps also decreases. SSD uses lower resolution layers to detect larger scale objects. For example, the 4×4 feature maps are used for larger scale object.

SSD adds 6 more auxiliary convolution layers after the VGG16. Five of them will be added for object detection. In three of those layers, we make 6 predictions instead of 4. In total, SSD makes 8732 predictions using 6 layers.

- **Default boundary box**

Just like Deep Learning, we can start with random predictions and use gradient descent to optimize the model. However, during the initial training, the model may fight with each other to determine what shapes (pedestrians or cars) to be optimized for which predictions. Empirical results indicate early training can be very unstable. The boundary box predictions below work well for one category but not for others. We want our initial predictions to be diverse and not looking similar. If our predictions cover more shapes, like the one below, our model can detect more object types. This kind of head start makes training much easier and more stable.

In real-life, boundary boxes do not have arbitrary shapes and sizes. Cars have similar shapes and pedestrians have an approximate aspect ratio of 0.41. In the KITTI dataset used in autonomous driving, the width and height distributions for the boundary boxes are highly clustered.

Conceptually, the ground truth boundary boxes can be partitioned into clusters with each cluster represented by a default boundary box (the centroid of the cluster). So, instead of making random guesses, we can start the guesses based on those default boxes.

To keep the complexity low, the default boxes are pre-selected manually and carefully to cover a wide spectrum of real-life objects. SSD also keeps the default boxes to a minimum (4 or 6) with one prediction per default box. Now, instead of using global coordination for the box location, the boundary box predictions are relative to the default boundary boxes at each cell (Δcx , Δcy , Δw , Δh), i.e. the offsets (difference) to the default box at each cell for its center (cx , cy), the width and the height.

For each feature map layers, it shares the same set of default boxes centered at the corresponding cell. But different layers use different sets of default boxes to customize object detections at different resolutions. The 4 green boxes below illustrate 4 default boundary boxes.

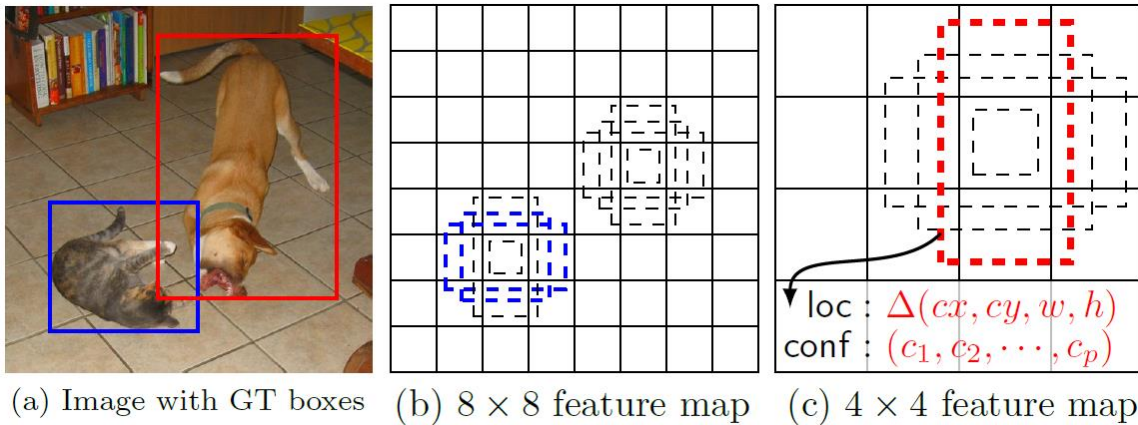
$$w = scale \cdot \sqrt{\text{aspect ratio}}$$

$$h = \frac{scale}{\sqrt{\text{aspect ratio}}}$$

Then SSD adds an extra default box with scale:

$$scale = \sqrt{scale \cdot \text{scale at next level}}$$

• Multi-scale feature maps & default boundary boxes



- After going through a certain of convolutions for feature extraction, we obtain a feature layer of size $m \times n$ (number of locations) with p channels, such as 8×8 or 4×4 above. And a 3×3 conv is applied on this $m \times n \times p$ feature layer.
- For each location, we got k bounding boxes. These k bounding boxes have different sizes and aspect ratios. The concept is, maybe a vertical rectangle is more fit for human, and a horizontal rectangle is more fit for car
- For each of the bounding box, we will compute c class scores and 4 offsets relative to the original default bounding box shape.

- Thus, we got (c+4) kmn outputs

Here is an example of how SSD combines multi-scale feature maps and default boundary boxes to detect objects at different scales and aspect ratios. The dog below matches one default box (in red) in the 4×4 feature map layer, but not any default boxes in the higher resolution 8×8 feature map. The cat which is smaller is detected only by the 8×8 feature map layer in 2 default boxes (in blue). Higher-resolution feature maps are responsible for detecting small objects. The first layer for object detection conv4_3 has a spatial dimension of 38×38 , a pretty large reduction from the input image. Hence, SSD usually performs badly for small objects comparing with other detection methods. If it is a problem, we can mitigate it by using images with higher resolution.

• Loss function

The localization loss is the mismatch between the ground truth box and the predicted boundary box. SSD only penalizes predictions from positive matches. We want the predictions from the positive matches to get closer to the ground truth. Negative matches can be ignored.

The localization loss between the predicted box l and the ground truth box g is defined as the smooth L1 loss with cx, cy as the offset to the default bounding box d of width w and height h .

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^N \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{smooth}_{L1}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \quad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \quad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

$$x_{ij}^p = \begin{cases} 1 & \text{if IoU} > 0.5 \text{ between default box } i \text{ and ground true box } j \text{ on class } p \\ 0 & \text{otherwise} \end{cases}$$

The confidence loss is the loss of making a class prediction. For every positive match prediction, we penalize the loss according to the confidence score of the corresponding class. For negative match predictions, we penalize the loss according to the confidence score of the class "0": class "0" classifies no object is detected.

It is calculated as the softmax loss over multiple classes confidences c (class score).

$$L_{conf}(x, c) = - \sum_{i \in Pos}^N x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in Neg} \log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

where N is the number of matched default boxes.

The final loss function is computed as:

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

where N is the number of positive matches and α is the weight for the localization loss.

• Hard negative mining

However, we make far more predictions than the number of objects present. So there are many more negative matches than positive matches. This creates a class imbalance that hurts training. We are training the model to learn background space rather than detecting objects. However, SSD still requires negative sampling so it can learn what constitutes a bad prediction. So, instead of using all the negatives, we sort those negatives by their calculated confidence loss. SSD picks the negatives with the top loss and makes sure the ratio between the picked negatives and positives is at most 3:1. This leads to faster and more stable training.

Inference time

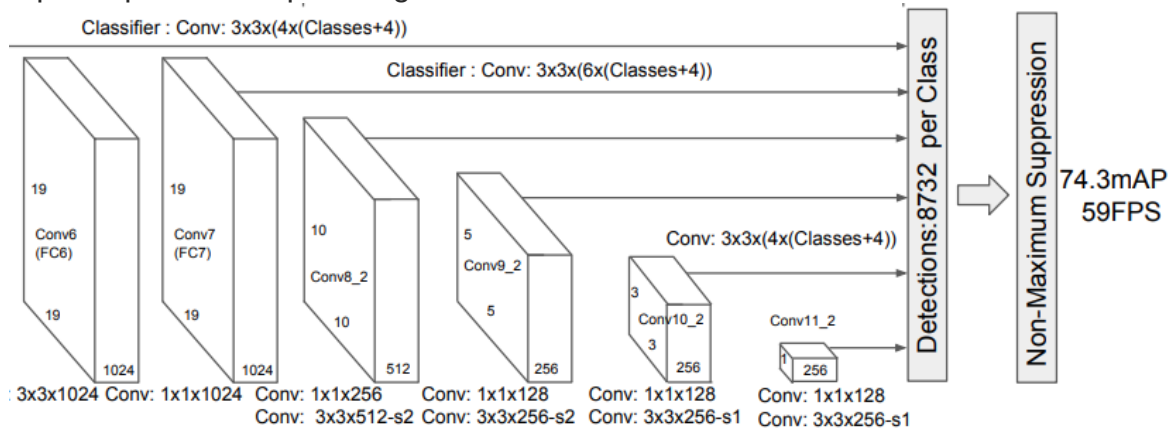
SSD makes many predictions (8732) for better coverage of location, scale, and aspect ratios, more than many other detection methods. However, many predictions contain no object. Therefore, any predictions with class confidence scores lower than 0.01 will be eliminated.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

SSD makes more predictions. Improvements allow SSD to use lower resolution images for similar accuracy.

SSD uses **non-maximum suppression** to remove duplicate predictions pointing to the same object. SSD sorts the predictions by the confidence scores. Start from the

top confidence prediction, SSD evaluates whether any previously predicted boundary boxes have an IoU higher than 0.45 with the current prediction for the same class. If found, the current prediction will be ignored. At most, we keep the top 200 predictions per image.



Result

The model is trained using SGD with an initial learning rate of 0.001, 0.9 momentum, 0.0005 weight decay, and batch size 32. Using an Nvidia Titan X on the VOC2007 test, SSD achieves 59 FPS with mAP 74.3% on the VOC2007 test, vs. Faster R-CNN 7 FPS with mAP 73.2% or YOLO 45 FPS with mAP 63.4%.

Here is the accuracy comparison for different methods. For SSD, it uses an image size of 300×300 or 512×512 .

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast [6]	07	66.9	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8
Fast [6]	07+12	70.0	77.0	78.1	69.3	59.4	38.3	81.6	78.6	86.7	42.8	78.8	68.9	84.7	82.0	76.6	69.9	31.8	70.1	74.8	80.4	70.4
Faster [2]	07	69.9	70.0	80.6	70.1	57.3	49.9	78.2	80.4	82.0	52.2	75.3	67.2	80.3	79.8	75.0	76.3	39.1	68.3	67.3	81.1	67.6
Faster [2]	07+12	73.2	76.5	79.0	70.9	65.5	52.1	83.1	84.7	86.4	52.0	81.9	65.7	84.8	84.6	77.5	76.7	38.8	73.6	73.9	83.0	72.6
Faster [2]	07+12+COCO	78.8	84.3	82.0	77.7	68.9	65.7	88.1	88.4	88.9	63.6	86.3	70.8	85.9	87.6	80.1	82.3	53.6	80.4	75.8	86.6	78.9
SSD300	07	68.0	73.4	77.5	64.1	59.0	38.9	75.2	80.8	78.5	46.0	67.8	69.2	76.6	82.1	77.0	72.5	41.2	64.2	69.1	78.0	68.5
SSD300	07+12	74.3	75.5	80.2	72.3	66.3	47.6	83.0	84.2	86.1	54.7	78.3	73.9	84.5	85.3	82.6	76.2	48.6	73.9	76.0	83.4	74.0
SSD300	07+12+COCO	79.6	80.9	86.3	79.0	76.2	57.6	87.3	88.2	88.6	60.5	85.4	76.7	87.5	89.2	84.5	81.4	55.0	81.9	81.5	85.9	78.9
SSD512	07	71.6	75.1	81.4	69.8	60.8	46.3	82.6	84.7	84.1	48.5	75.0	67.4	82.3	83.9	79.4	76.6	44.9	69.9	69.1	78.1	71.8
SSD512	07+12	76.8	82.4	84.7	78.4	73.8	53.2	86.2	87.5	86.0	57.8	83.1	70.2	84.9	85.2	83.9	79.7	50.3	77.9	73.9	82.5	75.3
SSD512	07+12+COCO	81.6	86.6	88.3	82.4	76.0	66.3	88.6	88.9	89.1	65.1	88.4	73.6	86.5	88.9	85.3	84.6	59.1	85.0	80.4	87.4	81.2

This is the recap of the speed performance in frame per second.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	$\sim 1000 \times 600$
Fast YOLO	52.7	155	1	98	448×448
YOLO (VGG16)	66.4	21	1	98	448×448
SSD300	74.3	46	1	8732	300×300
SSD512	76.8	19	1	24564	512×512
SSD300	74.3	59	8	8732	300×300
SSD512	76.8	22	8	24564	512×512

4.RESULTS AND DISCUSSION

4.1 RESULT:

The results were correct because to the combination of SSD architecture and the Mobile Net concept. The individual who was detected was encased in a bounding box, with the person's coordinates displayed above. Any low-power device, such as a drone, can operate the suggested model. The results were validated using a real-time video captured from a drone's camera attached to a Raspberry pi. To detect the corresponding images, the drone did not use an external GPU. The frame rate at which the next item is identified is expressed in frames per second (FPS). The bounding boxes that are shown on the discovered items are called boxes. We used the benefits of transfer learning to fine-tune our model for our project. As a result, the SSD 300 was the most practical type for usage with the Mobile Net, resulting in great accuracy. The results are depicted in Fig.2.

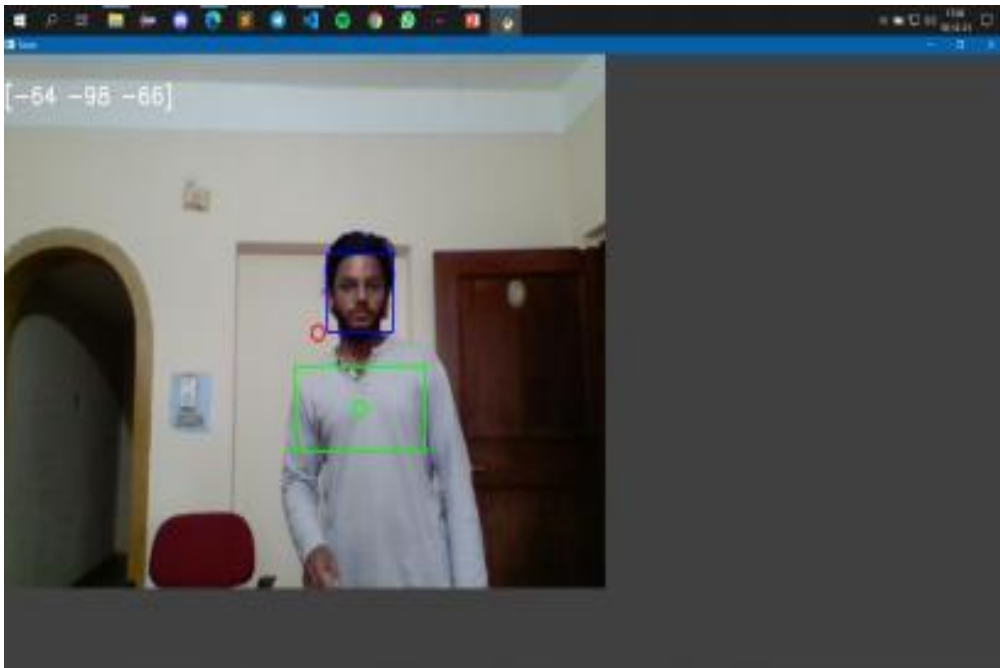


Fig.2 UAV detected co-ordinates and identified person

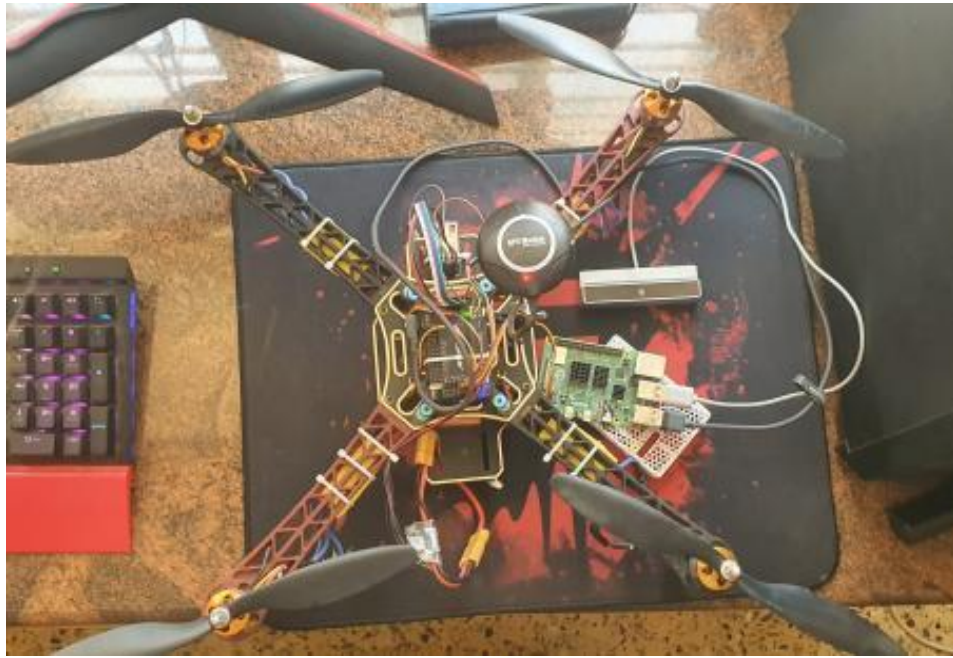


Fig.3 UAV

5. CONCLUSIONS

This describes a novel and practical applied vision and control pipeline for controlling autonomous quad copters in the task of following a person in a large empty field. While the system is usable in some applications right now it can be a great base for further work to especially improve the system response for fast movements. In order to do these improvements in the depth measuring hardware needs to be made. The single solid-state Lidar is often not pointed towards the person during flight returning unsuitable invalid data. These gaps in depth data makes it difficult for the control system to respond correctly.

REFERENCES

- [1] Development of UAV-Based Target Tracking and Recognition Systems, Shuaijun Wang, Fan Jiang, Bin Zhang, Rui Ma, and Qi Hao, Member, IEEE (2019)
- [2] Single Object Tracking Using Multi-Drones and an Agent Sharing Network Pengfei Zhu, Jiayu Zheng, Dawei Du, Longyin Wen, Yiming Sun, Qinghua Hu, Pengfei Zhu, Jiayu Zheng, Dawei Du, Longyin Wen, Yiming Sun, Qinghua Hu
- [3] Quad Rotorcraft Switching Control: An Application for the Task of Path Following Luis Rodolfo García Carrillo, Gerardo R. Flores Colunga, Guillaume Sanahuja, and Rogelio Lozano
- [4] Any Object Tracking and Following by a Flying Drone Roman Bartak and Adam Vyškovský Charles University in Prague, Faculty of Mathematics and Physics Malostranské náměstí 25, Praha 1, Czech Republic.
- [5] Object recognition and tracking using Haar-like Features Cascade Classifiers: Application to a quad-rotor UAV, Luis Arreola-Gesem Gudino and Gerardo Flores
- [6] Autonomous quad copter control for person tracking, Sieuwe Elferink
- [7] Drone-based real-time object tracking using multi-inertial sensing data Wei Zhu, Peng Chen, Yuanjie Dang, Ronghua Liang, Senior Member, IEEE, Peng Chen, Yuanjie Dang, Ronghua Liang, Senior Member, IEEE, Peng Chen, Yuanjie Dang, Ronghua Liang, Senior Member, IEEE, Wei Zhu, and Xiaofei He, Senior Member,
- [8]. B. A. G. de Oliveira, F. M. F. Ferreira, and C. A. P. da Silva Martins (2018). Object that is both quick and light.

Detection Network: Detection and recognition in resource constrained devices. IEEE Access, 6, 8714- 8724.

[9]. Real World Object Detection Dataset for Quadcopter Unmanned Aerial Vehicle Detection Maciej Pawełczyk, Marek Wojtyra Institute of Aeronautics and Applied Mechanics, Warsaw University of Technology, Warsaw, Poland Corresponding author: Maciej Pawełczyk.

[10]. EMBEDDED UAV VISUAL OBJECT DETECTION AND TRACKING IN REAL-TIME Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece, Paraskevi Nousi, Ioannis Mademlis, Iason Karakostas, Anastasios Tefas, Ioannis Pitas

[11]. Convolutional Neural Networks with Time Domain Motion Features for Drone Video Object Detection Yugui Zhang¹ Liuqing Shen¹ Xiaoyan Wang¹ Hai-Miao Hu^{1, 2} ¹ Beijing Key Laboratory of Digital Media, School of Computer Science and Engineering ² State Key Laboratory of Virtual Reality Technology and Systems Beihang University, Beijing, China

[12]. Deep Learning-Based Object Detection for Quadcopter Drones Widodo Budiharto¹, Alexander A S Gunawan², Jarot S. Suroso³, Andry Chowanda¹, Aurello Patrik¹, and Gaudi Utama¹ ¹Computer Science Department, Bina Nusantara University's School of Computer Science

[13]. Drone Detection and Tracking Based on Phase-Interferometric Doppler Radar Michael Jian, Zhenzhong Lu and Victor C. Chen Ancortek Inc. Fairfax, Virginia 22030 U.S.A.

[14]. S. Guennouni, A. Ahaitouf, and A. Mansouri (2014, October). Multiple object detection on an embedded device using OpenCV. In 2014, the IEEE International Colloquium on Information Science and Technology (CIST) held its third annual meeting (pp. 374-377). IEEE.

[15]. R. Simhambhatla, K. Okiah, S. Kuchkula, and R. Slater, R. Slater, R. Slater, R. Slater, R. Slater, R. Slater, R. Slater, R. (2019). Evaluation of Deep Learning Techniques for Object Detection under Different Driving Conditions in Self-Driving Cars 2(1), SMU Data Science Review, 23.

[16]. Chi-Tinh Dang; Hoang-The Pham; Thanh-Binh Pham; Nguyen-Vu

Truong: Vision based ground object tracking using AR.Drone quadrotor. In Proceedings of 2013 International Conference on Control, Automation and Information Sciences (ICCAIS), pp. 146–151, Nha Trang: IEEE (2013).

[17]. Kalal, Z.: Tracking Learning Detection. PhD thesis. University of Surrey, Faculty of Engineering and Physical Sciences, Centre for Vision, Speech and Signal Processing (2011)

[18]. King, M.: Process Control: A Practical Approach. Chichester, UK: John Wiley & Sons Ltd. (2010)

[19]. Matzner, F.: Tracking of 3D Movement. Bachelor Thesis. Charles University in Prague, Faculty of Mathematics and Physics (2014).

[20]. Szeliski, R.: Computer Vision: Algorithms and Applications. Springer Verlag (2010) Available from: <http://www.szeliski.org/Book>.

APPENDIX

A) SOURCE CODE:

obje.py

```
import time
import cv2 as cv
import numpy as np
import math

prototxt_path = "MobileNetSSD_deploy.prototxt.txt"
model_path = "MobileNetSSD_deploy.caffemodel"

CLASSES = [
    "background",
    "aeroplane",
    "bicycle",
    "bird",
    "boat",
    "bottle",
    "bus",
    "car",
    "cat",
    "chair",
    "cow",
    "diningtable",
    "dog",
    "horse",
    "motorbike",
    "person",
    "pottedplant",
    "sheep",
    "sofa",
    "train",
    "tvmonitor",
]

net = cv.dnn.readNetFromCaffe(prototxt_path, model_path)

def process_frame_MobileNetSSD(next_frame):
    rgb = cv.cvtColor(next_frame, cv.COLOR_BGR2RGB)
    (H, W) = next_frame.shape[:2]
```

```

blob = cv.dnn.blobFromImage(next_frame, size=(300, 300), ddepth=cv.CV_8U)
net.setInput(blob, scalefactor=1.0 / 127.5, mean=[127.5, 127.5, 127.5])
detections = net.forward()

```

```

for i in np.arange(0, detections.shape[2]):

```

```

    confidence = detections[0, 0, i, 2]

```

```

    if confidence > 0.7:

```

```

        idx = int(detections[0, 0, i, 1])

```

```

        if CLASSES[idx] != "person":
            continue

```

```

        box = detections[0, 0, i, 3:7] * np.array([W, H, W, H])
        (startX, startY, endX, endY) = box.astype("int")

```

```

        cv.rectangle(next_frame, (startX, startY), (endX, endY), (0, 255, 0), 3)

```

```

    return next_frame

```

```

def VehicheDetection_UsingMobileNetSSD():

```

```

    cap = cv.VideoCapture(0)

```

```

    fps = 20

```

```

    while True:

```

```

        ret, next_frame = cap.read()

```

```

        if ret == False:
            break

```

```

        next_frame = process_frame_MobileNetSSD(next_frame)

```

```

        # write frame

```

```

        cv.imshow("", next_frame)
        key = cv.waitKey(50)

```

```

        if key == 27:
            break

```

```
cap.release()
cv.destroyAllWindows()
```

VehicleDetection_UsingMobileNetSSD()

new_dist.py

```
import cv2
import numpy as np
import control
import time
import imutils

FPS = 25
# control.connect_drone("tcp:127.0.0.1:5762")
# control.configure_PID("PID")
cap = cv2.VideoCapture(0)
OVERRIDE = True
oSpeed = 5
S = 20
tDistance = 5
for_back_velocity = 0
left_right_velocity = 0
up_down_velocity = 0
faceSizes = [1026, 684, 456, 304, 202, 136, 90]
acc = [500, 250, 250, 150, 110, 70, 50]
dimensions = (960, 720)
UDOffset = 150
szX = 100
szY = 55
detector = cv2.CascadeClassifier("haarcascade_frontalface_alt2.xml")
recognizer = cv2.face.LBPHFaceRecognizer_create()
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 960)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
cap.set(cv2.CAP_PROP_FPS, FPS)

while True:

    # time.sleep(1 / FPS)

    k = cv2.waitKey(20)
    if k == ord("t"):
        print("Taking Off")
```

```

        control.arm_and_takeoff(3)
    if k == ord("l"):
        print("Landing")
        control.land()
    if k == 8:
        if not OVERRIDE:
            OVERRIDE = True
            print("OVERRIDE ENABLED")
        else:
            OVERRIDE = False
            print("OVERRIDE DISABLED")
    if k == 27:
        break

if OVERRIDE:
    # S & W to fly forward & back
    if k == ord("w"):
        for_back_velocity = int(S * oSpeed)
    elif k == ord("s"):
        for_back_velocity = -int(S * oSpeed)
    else:
        for_back_velocity = 0

    # a & d to pan left & right
    if k == ord("d"):
        yaw_velocity = 15
    elif k == ord("a"):
        yaw_velocity = -15
    else:
        yaw_velocity = 0

    # Q & E to fly up & down
    if k == ord("e"):
        up_down_velocity = int(S * oSpeed)
    elif k == ord("q"):
        up_down_velocity = -int(S * oSpeed)
    else:
        up_down_velocity = 0

    # c & z to fly left & right
    if k == ord("c"):
        left_right_velocity = int(S * oSpeed)
    elif k == ord("z"):
        left_right_velocity = -int(S * oSpeed)

```

```

else:
    left_right_velocity = 0

ok, image = cap.read()
image = imutils.resize(image, width=920, height=720)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
rects = detector.detectMultiScale(
    gray,
    scaleFactor=1.5,
)
tSize = faceSizes[tDistance]
cWidth = int(dimensions[0] / 2)
cHeight = int(dimensions[1] / 2)
cv2.circle(image, (cWidth, cHeight), 10, (0, 0, 255), 2)
# loop over the faces and draw a rectangle surrounding each
for (x, y, w, h) in rects:
    end_cord_x = x + w
    end_cord_y = y + h
    end_size = w * 2
    fbCol = (255, 0, 0)
    fbStroke = 2
    # these are our target coordinates
    targ_cord_x = int((end_cord_x + x) / 2)
    targ_cord_y = int((end_cord_y + y) / 2) + UDOffset
    vTrue = np.array((cWidth, cHeight, tSize))
    vTarget = np.array((targ_cord_x, targ_cord_y, end_size))
    vDistance = vTrue - vTarget
    cv2.rectangle(image, (x, y), (x + w, y + h), fbCol, 2)
    cv2.rectangle(
        image,
        (targ_cord_x - szX, targ_cord_y - szY),
        (targ_cord_x + szX, targ_cord_y + szY),
        (0, 255, 0),
        fbStroke,
    )
    cv2.circle(image, (targ_cord_x, targ_cord_y), 10, (0, 255, 0), 2)
    cv2.putText(
        image,
        str(vDistance),
        (0, 64),
        cv2.FONT_HERSHEY_SIMPLEX,
        1,
        (255, 255, 255),
    )

```

```

        2,
    )
    cv2.imshow("Faces", image)

```

```

cap.release()
cv2.destroyAllWindows()

```

ultrasonic.py

```

import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
#define ultrasonic sensor pin
PIN_TRIG_FRONT = 23
PIN_TRIG_BACK = 24
PIN_TRIG_LEFT = 25
PIN_TRIG_RIGHT = 8
PIN_ECHO_FRONT = 12
PIN_ECHO_BACK = 16
PIN_ECHO_LEFT = 20
PIN_ECHO_RIGHT = 21
#end

def measure_front():
    GPIO.setup(PIN_TRIG_FRONT,GPIO.OUT)
    GPIO.setup(PIN_ECHO_FRONT,GPIO.IN)
    GPIO.output(PIN_TRIG_FRONT,False)  #SET TO 0 OR FALSE TO
SETTLE
    time.sleep(0.2)
    GPIO.output(PIN_TRIG_FRONT,True)
    time.sleep(0.00001)
    GPIO.output(PIN_TRIG_FRONT,False)
    while GPIO.input(PIN_ECHO_FRONT)==0:
        pulse_start=time.time()
    while GPIO.input(PIN_ECHO_FRONT)==1:
        pulse_end=time.time()
    pulse_duration=pulse_end-pulse_start
    distance=pulse_duration*17150
    distance=round(distance,2)
    print("sensor front distance:",distance,"cm")
    time.sleep(0.5)
    return distance

def measure_back():
    GPIO.setup(PIN_TRIG_BACK,GPIO.OUT)
    GPIO.setup(PIN_ECHO_BACK,GPIO.IN)
    GPIO.output(PIN_TRIG_BACK,False)  #SET TO 0 OR FALSE TO SETTLE
    time.sleep(0.2)

```

```

GPIO.output(PIN_TRIG_BACK,True)
time.sleep(0.00001)
GPIO.output(PIN_TRIG_BACK,False)
while GPIO.input(PIN_ECHO_BACK)==0:
    pulse_start=time.time()
while GPIO.input(PIN_ECHO_BACK)==1:
    pulse_end=time.time()
pulse_duration=pulse_end-pulse_start
distance=pulse_duration*17150
distance=round(distance,2)
print("sensor back distance:",distance,"cm")
time.sleep(0.5)
return distance

def measure_left():
    GPIO.setup(PIN_TRIG_LEFT,GPIO.OUT)
    GPIO.setup(PIN_ECHO_LEFT,GPIO.IN)
    GPIO.output(PIN_TRIG_LEFT,False) #SET TO 0 OR FALSE TO SETTLE
    time.sleep(0.2)
    GPIO.output(PIN_TRIG_LEFT,True)
    time.sleep(0.00001)
    GPIO.output(PIN_TRIG_LEFT,False)
    while GPIO.input(PIN_ECHO_LEFT)==0:
        pulse_start=time.time()
    while GPIO.input(PIN_ECHO_LEFT)==1:
        pulse_end=time.time()
    pulse_duration=pulse_end-pulse_start
    distance=pulse_duration*17150
    distance=round(distance,2)
    print("sensor left distance:",distance,"cm")
    time.sleep(0.5)
    return distance

def measure_right():
    GPIO.setup(PIN_TRIG_RIGHT,GPIO.OUT)
    GPIO.setup(PIN_ECHO_RIGHT,GPIO.IN)
    GPIO.output(PIN_TRIG_RIGHT,False) #SET TO 0 OR FALSE TO
SETTLE
    time.sleep(0.2)
    GPIO.output(PIN_TRIG_RIGHT,True)
    time.sleep(0.00001)
    GPIO.output(PIN_TRIG_RIGHT,False)
    while GPIO.input(PIN_ECHO_RIGHT)==0:
        pulse_start=time.time()
    while GPIO.input(PIN_ECHO_RIGHT)==1:
        pulse_end=time.time()
    pulse_duration=pulse_end-pulse_start
    distance=pulse_duration*17150
    distance=round(distance,2)

```

```

print("sensor right distance:",distance,"cm")
time.sleep(0.5)
return distance

```

mav.py

```

import time
import math
from dronekit import connect, VehicleMode, LocationGlobalRelative,
Command, LocationGlobal
from pymavlink import mavutil
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--connect', default = "")
args = parser.parse_args()

connection_string = args.connect

#-----
#----- FUNCTIONS
#-----
#-- Define arm and takeoff
def arm_and_takeoff(altitude):

    while not vehicle.is_armable:
        print("waiting to be armable")
        time.sleep(1)

    print("Arming motors")
    vehicle.mode = VehicleMode("GUIDED")
    vehicle.armed = True

    while not vehicle.armed: time.sleep(1)

    print("Taking Off")
    vehicle.simple_takeoff(altitude)

    while True:
        v_alt = vehicle.location.global_relative_frame.alt
        print(">> Altitude = %.1f m"%v_alt)
        if v_alt >= altitude - 1.0:
            print("Target altitude reached")
            break
        time.sleep(1)

#-- Define the function for sending mavlink velocity command in body frame
def set_velocity_body(vehicle, vx, vy, vz):

```

"" Remember: vz is positive downward!!!

<http://ardupilot.org/dev/docs/copter-commands-in-guided-mode.html>

Bitmask to indicate which dimensions should be ignored by the vehicle
(a value of 0b0000000000000000 or 0b0000001000000000 indicates that
none of the setpoint dimensions should be ignored). Mapping:

bit 1: x, bit 2: y, bit 3: z,

bit 4: vx, bit 5: vy, bit 6: vz,

bit 7: ax, bit 8: ay, bit 9:

""

```
msg = vehicle.message_factory.set_position_target_local_ned_encode(
    0,
    0, 0,
    mavutil.mavlink.MAV_FRAME_BODY_NED,
    0b0000111111000111, #-- BITMASK -> Consider only the velocities
    0, 0, 0, #-- POSITION
    vx, vy, vz, #-- VELOCITY
    0, 0, 0, #-- ACCELERATIONS
    0, 0)
vehicle.send_mavlink(msg)
vehicle.flush()
```

```
def clear_mission(vehicle):
```

```
""
```

```
Clear the current mission.
```

```
""
```

```
cmds = vehicle.commands
vehicle.commands.clear()
vehicle.flush()
```

```
# After clearing the mission you MUST re-download the mission from the
vehicle
```

```
# before vehicle.commands can be used again
```

```
# (see https://github.com/dronekit/dronekit-python/issues/230)
```

```
cmds = vehicle.commands
cmds.download()
cmds.wait_ready()
```

```
def download_mission(vehicle):
```

```
""
```

```
Download the current mission from the vehicle.
```

```
""
```

```
cmds = vehicle.commands
cmds.download()
cmds.wait_ready() # wait until download is complete.
```

```
def get_current_mission(vehicle):
```

```
    """
```

```
    Downloads the mission and returns the wp list and number of WP
```

```
    Input:
```

```
        vehicle
```

```
    Return:
```

```
        n_wp, wpList
```

```
    """
```

```
    print("Downloading mission")
```

```
    download_mission(vehicle)
```

```
    missionList = []
```

```
    n_WP = 0
```

```
    for wp in vehicle.commands:
```

```
        missionList.append(wp)
```

```
        n_WP += 1
```

```
    return n_WP, missionList
```

```
def ChangeMode(vehicle, mode):
```

```
    while vehicle.mode != VehicleMode(mode):
```

```
        vehicle.mode = VehicleMode(mode)
```

```
        time.sleep(0.5)
```

```
    return True
```

```
def get_distance_metres(aLocation1, aLocation2):
```

```
    """
```

```
    Returns the ground distance in metres between two LocationGlobal objects.
```

```
    This method is an approximation, and will not be accurate over large  
distances and close to the
```

```
earth's poles. It comes from the ArduPilot test code:
```

```
https://github.com/diydrones/ardupilot/blob/master/Tools/autotest/common.py
```

```
    """
```

```
    dlat = aLocation2.lat - aLocation1.lat
```

```
    dlong = aLocation2.lon - aLocation1.lon
```

```
    return math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5
```

```
def distance_to_current_waypoint(vehicle):
```

```
    """
```

```
    Gets distance in metres to the current waypoint.
```

```
    It returns None for the first waypoint (Home location).
```

```
    """
```

```
    nextwaypoint = vehicle.commands.next
```

```

    if nextwaypoint==0:
        return None
    missionitem=vehicle.commands[nextwaypoint-1] #commands are zero
indexed
    lat = missionitem.x
    lon = missionitem.y
    alt = missionitem.z
    targetWaypointLocation = LocationGlobalRelative(lat,lon,alt)
    distancetopoint = get_distance_metres(vehicle.location.global_frame,
targetWaypointLocation)
    return distancetopoint

def bearing_to_current_waypoint(vehicle):
    nextwaypoint = vehicle.commands.next
    if nextwaypoint==0:
        return None
    missionitem=vehicle.commands[nextwaypoint-1] #commands are zero
indexed
    lat = missionitem.x
    lon = missionitem.y
    alt = missionitem.z
    targetWaypointLocation = LocationGlobalRelative(lat,lon,alt)
    bearing = get_bearing(vehicle.location.global_relative_frame,
targetWaypointLocation)
    return bearing

def get_bearing(my_location, tgt_location):
    """
    Aproximation of the bearing for medium latitudes and sort distances
    """
    dlat = tgt_location.lat - my_location.lat
    dlong = tgt_location.lon - my_location.lon

    return math.atan2(dlong,dlat)

def condition_yaw(heading, relative=False):
    """
    Send MAV_CMD_CONDITION_YAW message to point vehicle at a
    specified heading (in degrees).
    This method sets an absolute heading by default, but you can set the
    `relative` parameter
    to `True` to set yaw relative to the current yaw heading.
    By default the yaw of the vehicle will follow the direction of travel. After
    setting
    the yaw using this function there is no way to return to the default yaw
    "follow direction
    of travel" behaviour (https://github.com/diydrones/ardupilot/issues/2427)
    For more information see:
    http://copter.ardupilot.com/wiki/common-mavlink-mission-command-

```

```

messages-mav_cmd/#mav_cmd_condition_yaw
"""
if relative:
    is_relative = 1 #yaw relative to direction of travel
else:
    is_relative = 0 #yaw is an absolute angle
# create the CONDITION_YAW command using command_long_encode()
msg = vehicle.message_factory.command_long_encode(
    0, 0,      # target system, target component
    mavutil.mavlink.MAV_CMD_CONDITION_YAW, #command
    0,         #confirmation
    heading,   # param 1, yaw in degrees
    0,         # param 2, yaw speed deg/s
    1,         # param 3, direction -1 ccw, 1 cw
    is_relative, # param 4, relative offset 1, absolute angle 0
    0, 0, 0)   # param 5 ~ 7 not used
# send command to vehicle
vehicle.send_mavlink(msg)

```

```

def saturate(value, minimum, maximum):
    if value > maximum: value = maximum
    if value < minimum: value = minimum
    return value

```

```

def add_angles(ang1, ang2):
    ang = ang1 + ang2
    if ang > 2.0*math.pi:
        ang -= 2.0*math.pi

    elif ang < -0.0:
        ang += 2.0*math.pi
    return ang

```

```

#-----
#----- INITIALIZE
#-----
#-- Setup the commanded flying speed
gnd_speed = 8 # [m/s]
radius    = 80
max_lat_speed = 4
k_err_vel  = 0.2
n_turns    = 3
direction  = 1 # 1 for cw, -1 ccw

```

```

mode      = 'GROUND'

```

```

#-----
#----- CONNECTION
#-----

```

```

#-- Connect to the vehicle
print('Connecting...')
vehicle = connect(connection_string)
#vehicle = connect('tcp:127.0.0.1:5762', wait_ready=True)

#-----
#----- MAIN FUNCTION
#-----
while True:

    if mode == 'GROUND':
        #--- Wait until a valid mission has been uploaded
        n_WP, missionList = get_current_mission(vehicle)
        time.sleep(2)
        if n_WP > 0:
            print ("A valid mission has been uploaded: takeoff!")
            mode = 'TAKEOFF'

    elif mode == 'TAKEOFF':
        time.sleep(1)
        #-- Takeoff
        arm_and_takeoff(5)

        #-- Change mode, set the ground speed
        vehicle.groundspeed = gnd_speed
        mode = 'MISSION'
        vehicle.commands.next = 1

        vehicle.flush()

        #-- Calculate the time for n_turns
        time_flight = 2.0*math.pi*radius/gnd_speed*n_turns
        time0 = time.time()

        print ("Switch mode to MISSION")

    elif mode == 'MISSION':
        #-- We command the velocity in order to maintain the vehicle on track
        #- vx = constant
        #- vy = proportional to off track error
        #- heading = along the path tangent

        my_location = vehicle.location.global_relative_frame
        bearing = bearing_to_current_waypoint(vehicle)
        dist_2_wp = distance_to_current_waypoint(vehicle)

        try:
            print("bearing %.0f dist = %.0f"%(bearing*180.0/3.14, dist_2_wp))

```

```

        heading = add_angles(bearing, -direction*0.5*math.pi)
        #print heading*180.0/3.14
        condition_yaw(heading*180/3.14)

        v_x    = gnd_speed
        v_y    = -direction*k_err_vel*(radius - dist_2_wp)
        v_y    = saturate(v_y, -max_lat_speed, max_lat_speed)
        print ("v_x = %.1f  v_y = %.1f"%(v_x, v_y))
        set_velocity_body(vehicle, v_x, v_y, 0.0)

    except Exception as e:
        print(e)

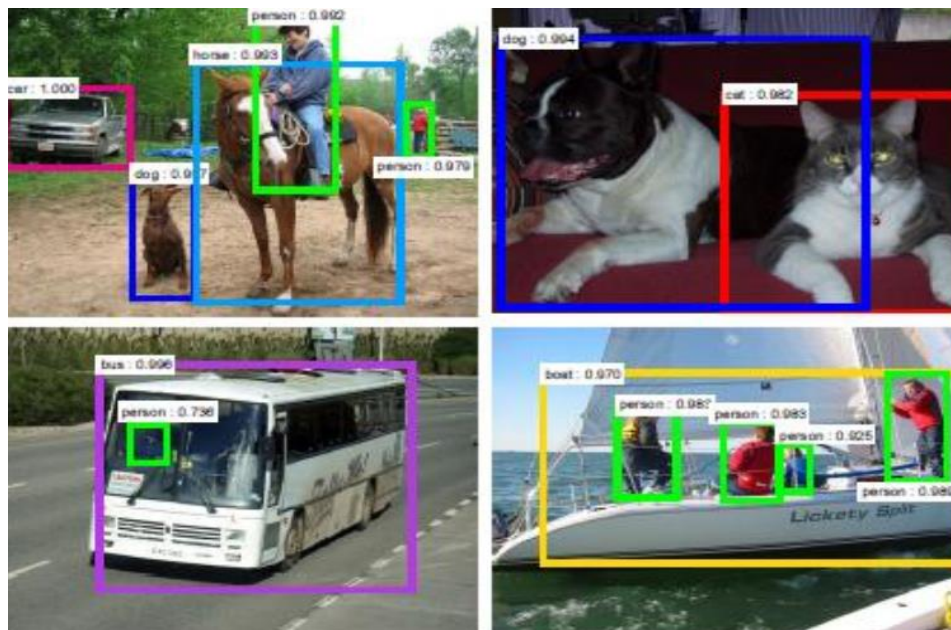
    if time.time() > time0 + time_flight:
        ChangeMode(vehicle, 'RTL')
        clear_mission(vehicle)
        mode = 'BACK'
        print (">> time to head Home: switch to BACK")

    elif mode == "BACK":
        if vehicle.location.global_relative_frame.alt < 1:
            print (">> Switch to GROUND mode, waiting for new missions")
            mode = 'GROUND'

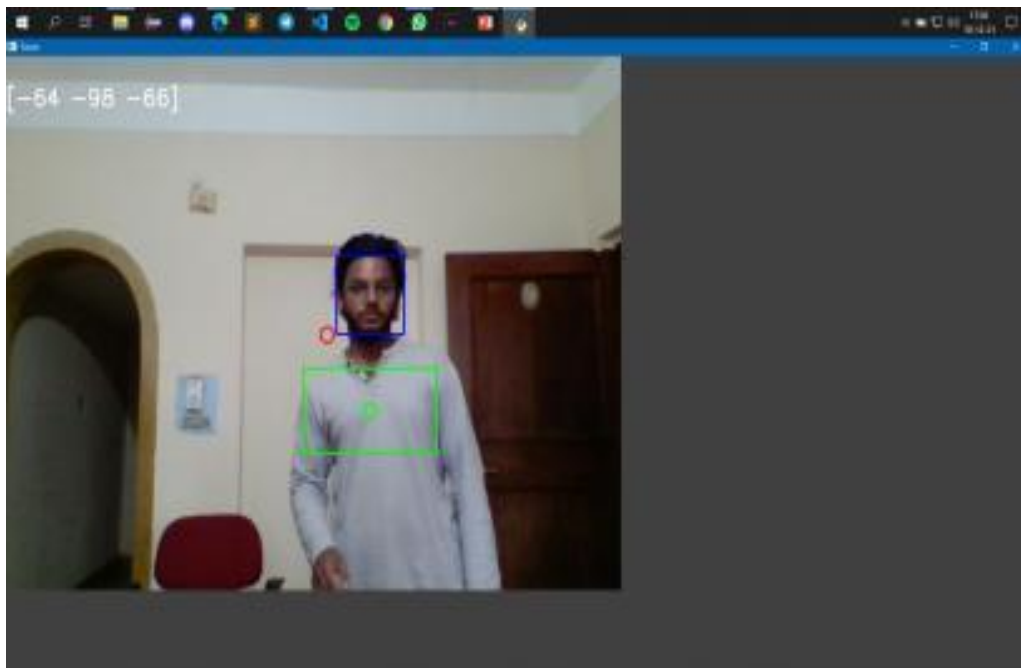
    time.sleep(0.5)

```

B) SCREENSHOTS:



Object detected successfully



Person tracking through UAV

C) PUBLICATION WITH PLAGIARISM REPORT:

KL32

by Green Official

Submission date: 16-Mar-2022 06:39AM (UTC-0700)

Submission ID: 1785606704

File name: OBJECT_DETECTION_AND_PERSON_TRACKING_USING_UAV.doc (3.96M)

Word count: 2421

Character count: 13515

KL32

ORIGINALITY REPORT

7%

SIMILARITY INDEX

4%

INTERNET SOURCES

5%

PUBLICATIONS

0%

STUDENT PAPERS

PRIMARY SOURCES

1

www.groundai.com

Internet Source

2%

2

Kumar Kunal, Md. Azhar Hussain, N Srinivasan, J. Albert Mayan. "Smart Irrigation and Tank Monitoring System", IOP Conference Series: Materials Science and Engineering, 2019

Publication

1%

3

Roman Bartak, Adam Vykovsky. "Any Object Tracking and Following by a Flying Drone", 2015 Fourteenth Mexican International Conference on Artificial Intelligence (MICA), 2015

Publication

1%

4

Peng Chen, Yuanjie Dang, Ronghua Liang, Wei Zhu, Xiaofei He. "Real-Time Object Tracking on a Drone With Multi-Inertial Sensing Data", IEEE Transactions on Intelligent Transportation Systems, 2018

Publication

1%

5	M. Jalasri, L. Lakshmanan. "A Survey: Integration of IoT and Fog Computing", 2018 Second International Conference on Green Computing and Internet of Things (ICGCIoT), 2018 Publication	1 %
6	Paraskevi Nousi, Ioannis Mademlis, Iason Karakostas, Anastasios Tefas, Ioannis Pitas. "Embedded UAV Real-Time Visual Object Detection and Tracking", 2019 IEEE International Conference on Real-time Computing and Robotics (RCAR), 2019 Publication	<1 %
7	Shuaijun Wang, Fan Jiang, Bin Zhang, Rui Ma, Qi Hao. "Development of UAV-Based Target Tracking and Recognition Systems", IEEE Transactions on Intelligent Transportation Systems, 2019 Publication	<1 %
8	dblp.uni-trier.de Internet Source	<1 %





SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

[DEEMED TO BE UNIVERSITY]

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in



National Conference on Computational Intelligence and Communication Networks



NCCICN

24TH – 25TH MARCH 2022



Certificate of Presentation

This is to certify that Dr./Mr./Ms. MANJUNATHA INTI of Sathyabama Institute of Science and Technology, has presented a paper entitled Object Detection and Person Tracking using UAV, in the National Conference on Computational Intelligence and Communication Networks (NCCICN 2022).

Dr. T. Sasikala

Conference Chair, Professor & Dean
School of Computing

Dr. L. Lakshmanan

Convener
Professor & Head, CSE

Dr. S. Vigneshwari

Convener
Professor & Head, CSE



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
[DEEMED TO BE UNIVERSITY]

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in



National Conference on Computational Intelligence and Communication Networks



NCCICN

24TH – 25TH MARCH 2022



Certificate of Presentation

*This is to certify that Dr./Mr./Ms. MANOJ PAWAR SJ of Sathyabama Institute of Science and Technology, has presented a paper entitled **Object Detection and Person Tracking using UAV**, in the National Conference on Computational Intelligence and Communication Networks (NCCICN 2022).*

Dr. T. Sasikala
Conference Chair, Professor & Dean
School of Computing

Dr. L. Lakshmanan
Convener
Professor & Head, CSE

Dr. S. Vigneshwari
Convener
Professor & Head, CSE