Internship Report

On

"UAV Detection using various techniques"

At

Armament Research and Development Establishment

Defence Research and Development Organization

Dr Homi Bhabha Road, Armament Post Pashan, Pune-411021

in partial fulfillment of the degree

Bachelor of Technology

in

INSTRUMENTATION AND CONTROL ENGINEERING

Vishwakarma Institute of Technology, Pune

Index – List of Content

CHAPTER 1

1.1 Introduction

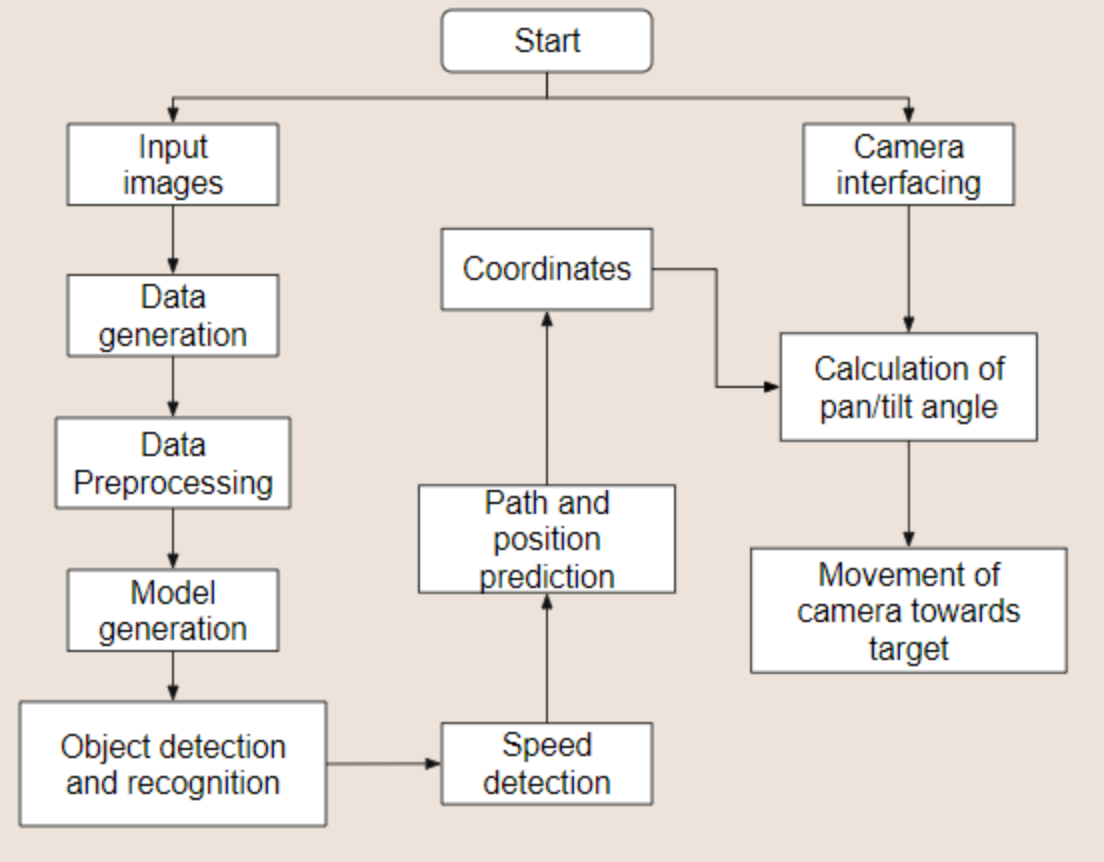
The fact that data is expanding daily by enormous numbers in today's fast-expanding globe, where all digital businesses are evolving at a great pace, is an unavoidable fact. The amount of data grows too large over time to remove any unmeaningful information. Huge data makes generalization challenging because there is a higher likelihood of variation from the same type of data. This data needs to be cleaned in order for it to be usable for making predictions about the future.

Machine Learning which is a subset of Artificial Intelligence (AI) comes to the rescue when dealing with huge amounts of data. With huge amounts of data comes the unprecedented errors while visualizing the data which need to be cleaned. For this data

preprocessing is used. It is a technique in which errors like null values, duplicate values, etc. are removed. Now that we have removed the data which was not very helpful for us, we have to sometimes scale it to normalize the values for acquiring more model accuracy. A very crucial aspect for building a machine learning model is the data itself.

Data generation in our project has been done through the augmentation of the existing data. Noise is the unwanted background objects that come with the pictures which can lead to inaccuracy. Let's move on to the most important part of our project. We would now be building a Deep Convolutional Neural Network to train our model and will be saving it for object detection & recognition , future predictions Further using it for speed detection.

1.2 Flowchart of the Project



Chapter 2

Data Collection

2.1 Collecting data via Kaggle & Roboflow

Real-world raw data and images are often incomplete, inconsistent and lacking in certain behaviors or trends. They are also likely to contain many errors. So, once collected, they are pre-processed into a format the machine learning algorithm can use for the model

Collecting data for training the ML model is the basic step in the machine learning pipeline. The predictions made by ML systems can only be as good as the data on which they have been trained.

We collected 5 types of objects which include Aeroplane, Birds, Helicopters, Drones, Ships of various categories. About 1000 Approx. images were used for the dataset to instruct the model.

We have used Roboflow for labeling of objects and using this labeled data to increase the precision of the model.

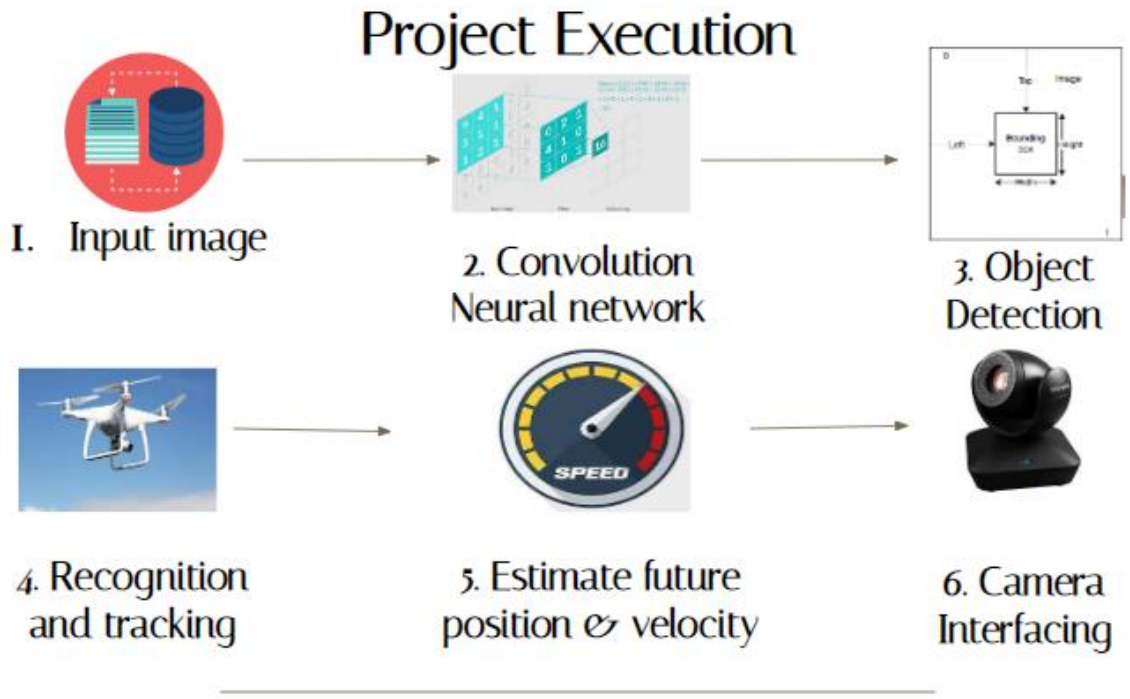
2.1.1 Pictures for dataset


نقلا عن التلفزيون المصري
الحياة
HD

Chapter 3

Implementation Steps

3.1 Introduction



Algorithms used are:

1. **YOLO**
2. **Optical flow**
3. **Background subtraction**
4. **Kalman filter**

All these algorithms are discussed in brief in the following document.

Chapter 4

YOLO(You Only Look Once)

4.1 Introduction

You only look once (YOLO) is a state-of-the-art, real-time object detection system. YOLOv4 is extremely fast and accurate. In mAP measured at .5 IOU YOLOv4 is on par with Focal Loss but about 4x faster. Moreover, you can easily trade off between speed and accuracy simply by changing the size of the model, no retraining.

4.1.1 How It Works

Prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections.

We use a totally different approach. We apply a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

Real-Time Detection on a Webcam

YOLO will display the current FPS and predicted classes as well as the image with bounding boxes drawn on top of it. You will need a webcam connected to the computer that OpenCV can connect to or it won't work.

YOLO stands for You Only Look Once, which is a real-time object detection system. We have use a pre-trained YOLO model to detect objects in images.

YOLO predicts the coordinates of bounding boxes directly using fully connected layers on top of the convolutional feature extractor. Predicting offsets instead of coordinates simplifies the problem and makes it easier for the network to learn.In addition to increased accuracy in predictions and a better Intersection over Union in bounding boxes (compared to real-time object detectors), YOLO has the inherent advantage of speed.

4.2 **Implementation**

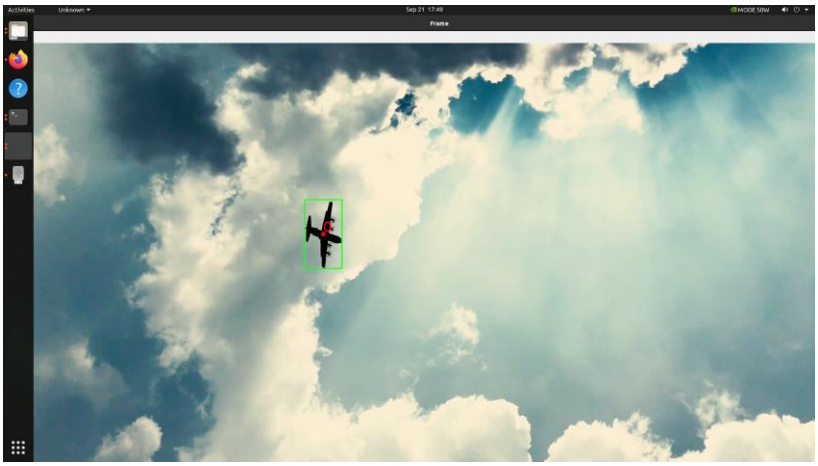


Figure: Center coordinate of object

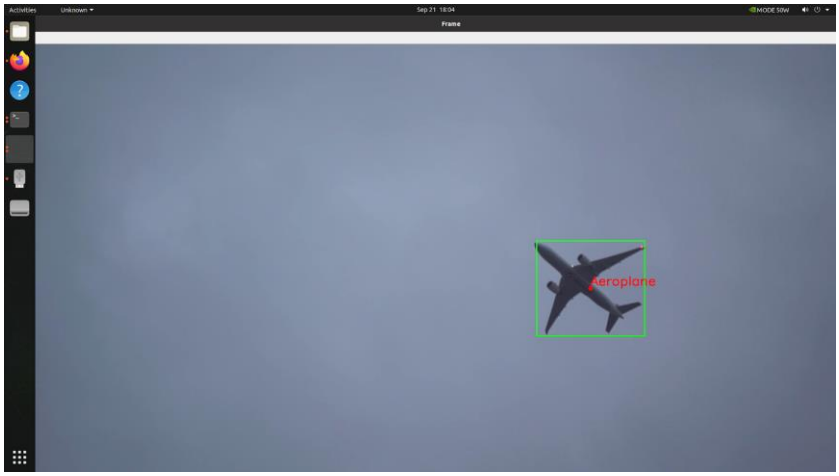


Figure: Object detection and recognition

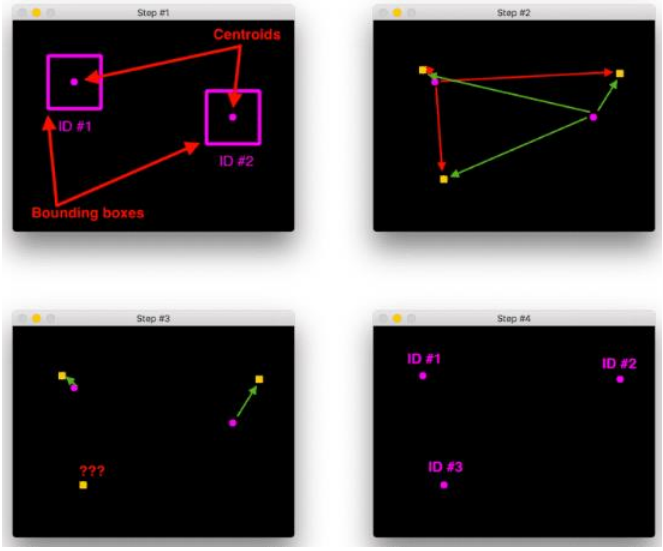


Figure: Centroid tracking

Step 1: Accept bounding box coordinates and compute centroids
Step 2: Compute Euclidean distance between new bounding boxes and existing objects
Step 3: Update *(x, y)*-coordinates of existing objects
Step 4: Register new objects
Step 5: Deregister old objects

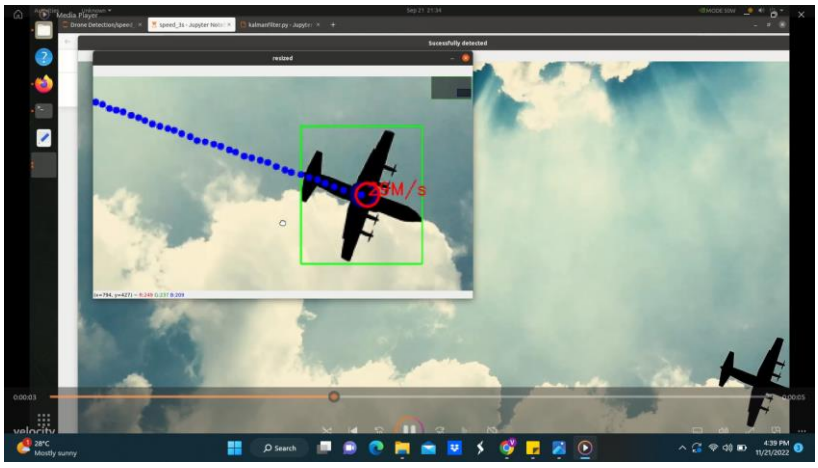


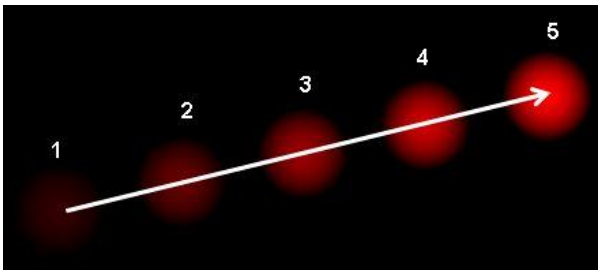
Figure: Speed estimation of object

Chapter 5

Optical Flow

5.1 Introduction

Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of an object or camera. It is a 2D vector field where each vector is a displacement vector showing the movement of points from first frame to second.



It shows a ball moving in 5 consecutive frames. The arrow shows its displacement vector. Optical flow has many applications in areas like :

- Structure from Motion
- Video Compression
- Video Stabilization ...

Optical flow works on several assumptions:

1. The pixel intensities of an object do not change between consecutive frames.
2. Neighboring pixels have similar motion.

5.1.1 Types of optical flow

There are two types of Optical Flow, and the first one is called Sparse Optical Flow. It computes the motion vector for the specific set of objects (for example – detected corners on image). Hence, it requires some preprocessing to extract features from the image, which will be the basement for the Optical Flow calculation. OpenCV provides some algorithm implementations to solve the Sparse Optical Flow task:

- Pyramid Lucas-Kanade
- Sparse RLOF

Using only a sparse feature set means that we will not have the motion information about pixels that are not contained in it. This restriction can be lifted using Dense Optical Flow algorithms which are supposed to calculate a motion vector for every pixel in the image. Some Dense Optical Flow algorithms are already implemented in OpenCV:

- Dense Pyramid Lucas-Kanade
- Farneback
- PCAFlow
- SimpleFlow
- RLOF
- DeepFlow
- DualTVL1

Sparse optical flow

Lucas-Kanade method

The Lucas-Kanade method is commonly used to calculate the Optical Flow for a sparse feature set. The main idea of this method is based on a local motion constancy assumption, where nearby pixels have the same displacement direction. This assumption helps to get the approximated solution for the equation with two variables.

We have seen an assumption before, that all the neighboring pixels will have similar motion. Lucas-Kanade method takes a 3x3 patch around the point. So all the 9 points have the same motion. We can find these 9 points. So now our problem becomes solving 9 equations with two unknown variables which are over-determined. A better solution is obtained with the least square fit method. Below is the final solution which is two equations-two unknown problems and solve to get the solution.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i {f_{x_i}}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i {f_{y_i}}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix}$$

(Check the similarity of the inverse matrix with Harris corner detector. It denotes that corners are better points to be tracked.)

So from the user point of view, the idea is simple, we give some points to track, and we receive the optical flow vectors of those points. But again there are some problems. Until now, we were dealing with small motions, so it fails when there is a large motion. To deal with this we use pyramids. When we go up in the pyramid, small motions are removed and large motions become small motions. So by applying Lucas-Kanade there, we get optical flow along with the scale.

Lucas-Kanade algorithm improvements

The Optical Flow algorithms really suffer from abrupt movements due to the algorithm's limitations. The common approach in practice is to use a multi-scaling trick. We need to create a so-called image pyramid, where every next image will be bigger than the previous one by some scaling factor (for example scale factor is 2). In terms of the fixed-size window, the abrupt movement on the small-sized image is more noticeable rather than on the large one. The founded displacement vectors in the small images will be used on the next larger pyramid stages to achieve better results.

Dense Optical Flow algorithms calculate the motion vector for the sparse feature set, so the common approach here is to use the Shi-Tomasi corner detector. It is used to find corners in the image and then calculate the corners' motion vector between two consecutive frames.

Dense optical flow

The dense optical flow algorithm output can be encoded as the HSV color scheme. Using the cv2.cartToPolar function, we can convert the displacement coordinates (dx, dy) into polar coordinates as magnitude and angle for every pixel. Here we can encode angle and magnitude as Hue and Value respectively, while Saturation remains constant. To show the optical flow properly, we need to convert the HSV into BGR format.

5.2 Implementation

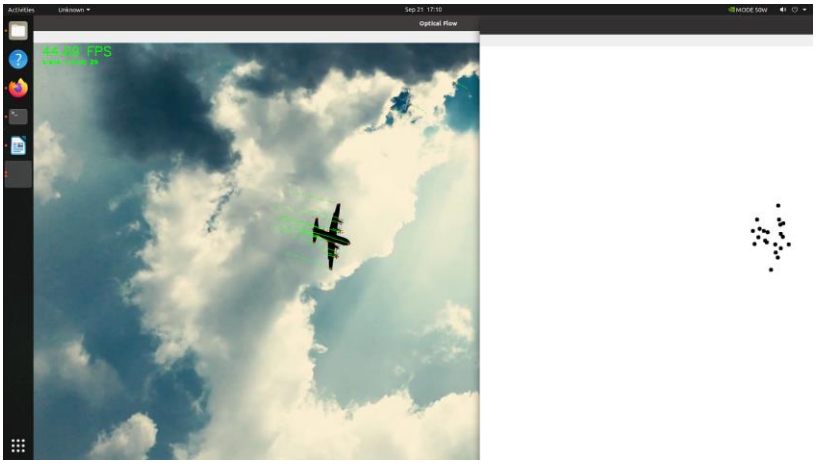


Figure: Sparse Optical flow

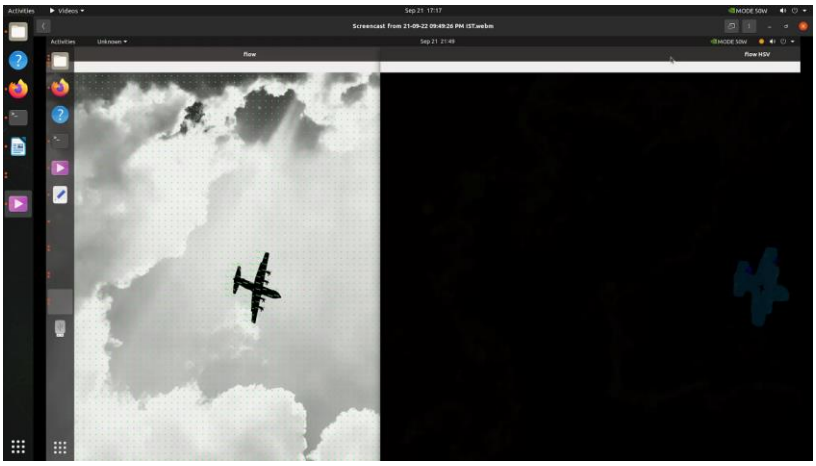


Figure: Dense Optical flow

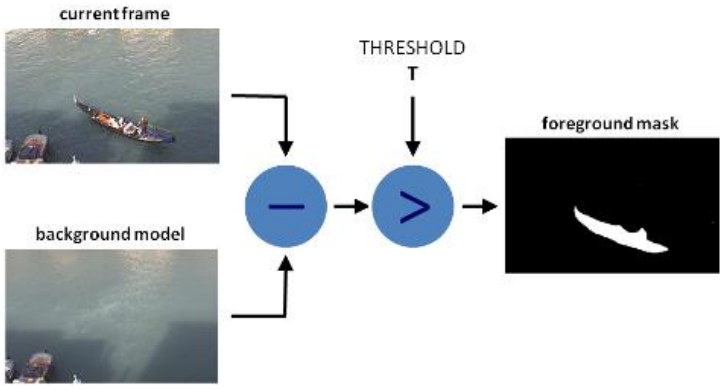
Chapter 6

Background Subtraction

6.1 Introduction

Background subtraction (BS) is a common and widely used technique for generating a foreground mask (namely, a binary image containing the pixels belonging to moving objects in the scene) by using static cameras.

As the name suggests, BS calculates the foreground mask performing a subtraction between the current frame and a background model, containing the static part of the scene or, more in general, everything that can be considered as background given the characteristics of the observed scene.



Background modeling consists of two main steps:

1) Background Initialization;
2) Background Update.

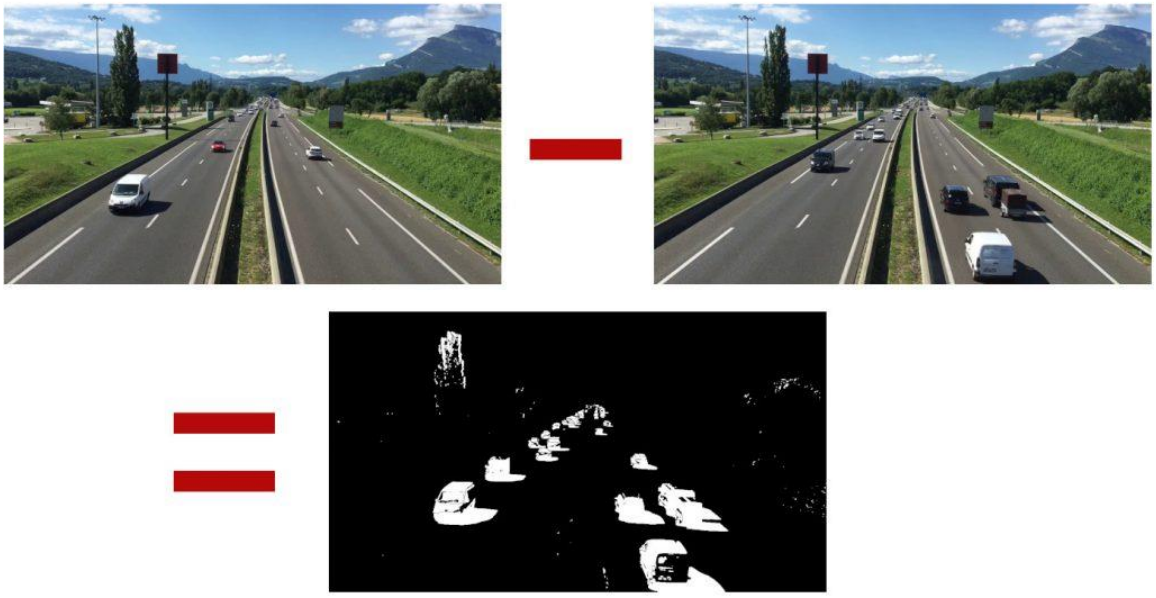
In the first step, an initial model of the background is computed, while in the second step that model is updated in order to adapt to possible changes in the scene.

6.1.1 Types of Background Subtraction

1) Manual Background Subtraction

A manual way which consists of taking the first frame and from that one subtracting each time the following frames from the first one.

In this way it is possible to distinguish the stable background from the objects that are moving. This method only works with a stable camera and a stable background.



2) MOG2 Background Subtractor

Background subtraction can be performed using MOG2 Background Subtraction nodes. The node needs a cvMat input, which is an OpenCV variable type designed for computer vision. MOG2 Background Subtraction has just 4 inputs:

- **Source** is the cvMat input on which you want to perform MOG2 Background Subtraction analysis
- **Learn rate** is the learning rate that the algorithm uses. Any negative value means an automatic learning rate, which is what you'll want to use in most cases.
- If **Blur** input is true and **Blur & Threshold** is false, then the output won't be thresholded, just blurred. If both are false, then neither will apply.
- If **Blur & Threshold** is true, then the result is first blurred and then thresholded.

The SubtractorMog2 has the advantage of working with a frame history, it works by default with the last 120 frames.

6.2 Implementation

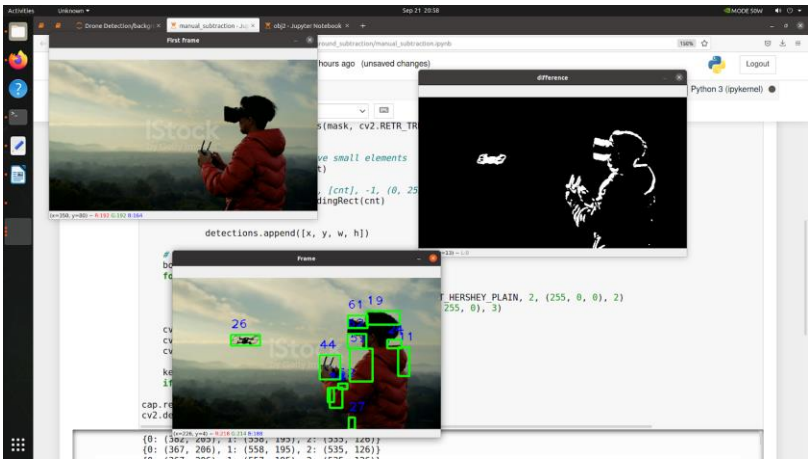


Figure: Manual Background Subtraction

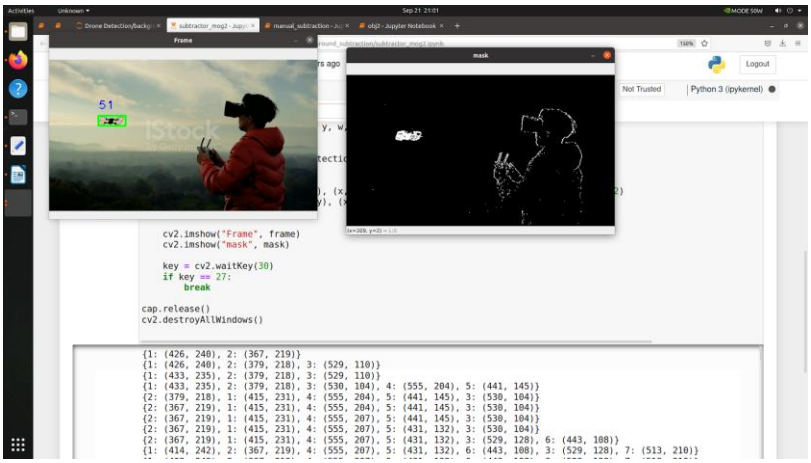


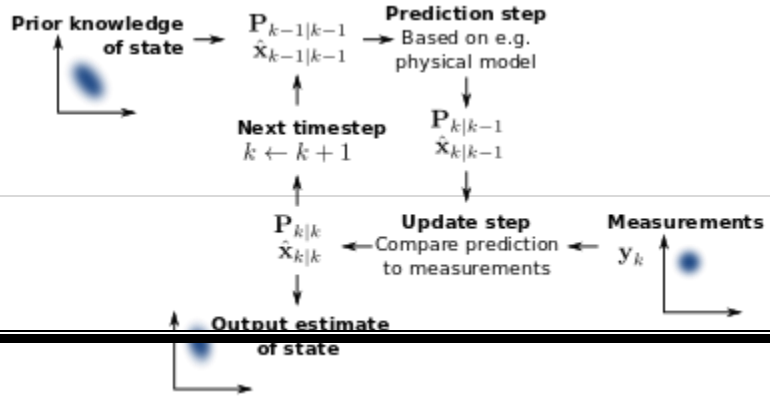
Figure: Subtractor MoG2

Chapter 7

Kalman Filter

7.1 Introduction

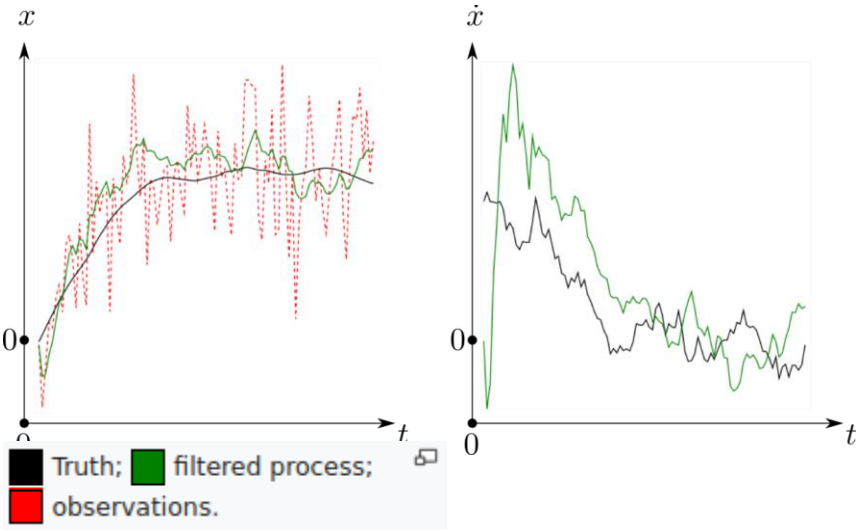
The Kalman filter is an algorithm that takes measurements over time and creates a prediction of the next measurements. This is used in many fields such as sensors, GPS, to predict the position in case of signal loss for a few seconds and this is what we will also see in computer vision.



The Kalman filter is a recursive estimator. This means that only the estimated state from the previous time step and the current measurement are needed to compute the estimate for the current state. In contrast to batch estimation techniques, no history of observations and/or estimates is required. In what follows, the notation x ^ n | m represents the estimate of x at time *n* given observations up to and including at time $m \leq n$.

The algorithm structure of the Kalman filter resembles that of Alpha beta filter. The Kalman filter can be written as a single equation; however, it is most often conceptualized as two distinct phases: "Predict" and "Update". The predict phase uses the state estimate from the previous timestep to produce an estimate of the state at the current timestep. This predicted state estimate is also known as the *a priori* state estimate because, although it is an estimate of the state at the current timestep, it does not include observation information from the current timestep. In the update phase, the innovation (the pre-fit residual), i.e. the difference between the current *a priori* prediction and the current observation information, is multiplied by the optimal Kalman gain and combined with the previous state estimate to refine the state estimate. This improved estimate based on the current observation is termed a *posteriori* state estimate.

Typically, the two phases alternate, with the prediction advancing the state until the next scheduled observation, and the update incorporating the observation. However, this is not necessary; if an observation is unavailable for some reason, the update may be skipped and multiple prediction procedures performed. Likewise, if multiple independent observations are available at the same time, multiple update procedures may be performed.



Extended Kalman Filter

In the extended Kalman filter (EKF), the state transition and observation models need not be linear functions of the state but may instead be nonlinear functions. These functions are of differentiable type.

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k$$
$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k$$

The function *f* can be used to compute the predicted state from the previous estimate and similarly the function *h* can be used to compute the predicted measurement from the predicted state. However, *f* and *h* cannot be applied to the covariance directly. Instead a matrix of partial derivatives (the Jacobian) is computed.

At each timestep the Jacobian is evaluated with current predicted states. These matrices can be used in the Kalman filter equations. This process essentially linearizes the nonlinear function around the current estimate.

Unscented Kalman filter

When the state transition and observation models—that is, the predict and update functions f and h are highly nonlinear, the extended Kalman filter can give particularly poor performance.This is because the covariance is propagated through linearization of the underlying nonlinear model. The unscented Kalman filter (UKF) uses a deterministic sampling technique known as the unscented transformation (UT) to pick a minimal set of sample points (called sigma points) around the mean. The sigma points are then propagated through the nonlinear functions, from which a new mean and covariance estimate are then formed. In addition, this technique removes the requirement to explicitly calculate Jacobians, which for complex functions can be a difficult

task in itself.

7.2 Implementation

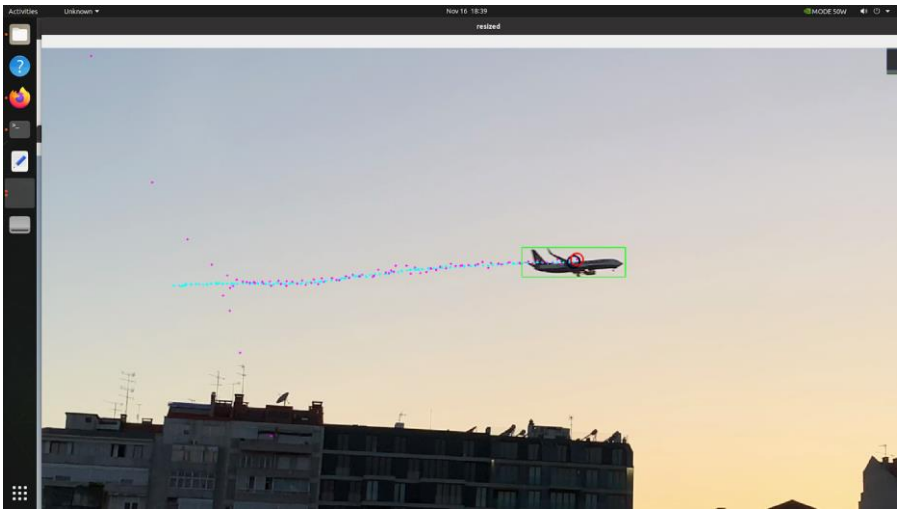


Figure: Predicting future position and tracking path of the object

7.3 Tracking Algorithms

Object tracking is a popular method because OpenCV has so many algorithms built-in that are specifically optimized for the needs and objectives of object or motion tracking. Specific Opencv object trackers include the BOOSTING, MIL, KCF, CSRT, MedianFlow, TLD, MOSSE, and GOTURN trackers. Each of these trackers is best for different goals. For example, CSRT is best when the user requires a higher object tracking accuracy and can tolerate slower FPS throughput. The selection of an OpenCV object tracking algorithm depends on the advantages and disadvantages of that specific tracker and the benefits: The KCF tracker is not as accurate compared to the CSRT but provides a comparably higher FPS. The MOSSE tracker is very fast, but its accuracy is even lower than tracking with KCF. Still, if you are looking for the fastest object tracking OpenCV method, MOSSE is a good choice. The GOTURN tracker is the only detector for deep learning based object tracking with OpenCV. The original implementation of GOTURN is in Caffe, but it has been ported to the OpenCV Tracking API.

Implementation results of Tracking algorithms:

1. **BOOSTING**-weak classifier to “focus” on their detection,relatively low speed

2. **MIL (Multiple instance Learning)** -more robust to noise, shows fairly good accuracy

3. **KCF (Kernelized Correlation filter)**-accuracy, stops tracking when the tracked object is lost

4. **CSRT (**Discriminative Correlation Filter with Channel and Spatial Reliability) -comparatively better accuracy and the best results of all

5. **MedianFlow**- speed of its movement is not too high

6. **TLD**- relatively good results

7. **MOSSE**- successful in continuing tracking the object if it was lost

8. **GOTURN**- Generic Object Tracking Using Regression Network) Tracker- good resistance to noise and obstructions.

Chapter 8

Camera Interfacing

Marshall CV610-UB

The Marshall CV610-U2 acts as a UVC command slave to Teleconference, U.C. or Video Capture Software programs that use it as a USB capture device. Once selected as an available camera, these software platforms send commands to camera as to compatible resolutions and frame rates based on bandwidth, software parameters, and/or computer capabilities. Some software platforms allow for direct adjust commands to change resolution and frame-rates, such as VLC Player and others. CV610-U2 adjusts as commanded by software platform or to the nearest resolution settings available (1080p, 720p, 480p). There is no way to manually adjust resolutions and frame-rates from the camera or camera OSD menu, since it relies on UVC (USB video class) commands. CV610-U2 adheres to UVC1.5 protocol Standard

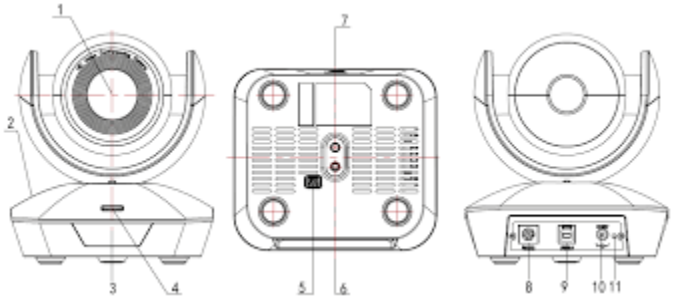
CV610-U2 SPECIFICATIONS

Video Format	MJPG - 1080P30, 720P30, 640*480P30
Video Port	USB2.0
Sensor	1/2.8 inch high quality 5MP CMOS sensor
Lens	F4.7~47.mm(10X), F1.8 – 14, Field of view: 62.5°(wide)-6.43°(tele)
Pan/tilt Rotation	Pan:±170°; Tilt:-30°~+90°, support up-side down installation
Pan/tilt Speed	Pan: 0.1°-120°/s; Tilt: 0.1°-80°/s
Preset	10 via IR remote setting, 128 via VISCA control, preset accuracy :0.1°
Control Port	RS232, RS485, USB2.0
Min. Lux	0.01lux
White Balance	Auto/Manual
Focus	Auto/Manual
Iris	Auto/Manual
Shutter	Auto/Manual
WDR	Supported
BLC	Supported
2D Noise Reduction	Supported
3D Noise Reduction	Supported
Input Voltage	DC12V
Dimension	148mm×132mm×161mm
Net weight	0.9KG (2LBS)

Formulae

Pan- (X /pixel Value)* camera Range

Tilt- (Y /pixel Value)* camera Range



Chapter 9

Conclusion

I had the chance to work on an industry project, engage with technological specialists, and contribute to the system within my five-month internship. The team made sure I have resources to grasp the concept, learn about real-life implementation and gradually take on assignments in line with my skills even though it was my first experience with the Industrial Project.

It helped me to better understand how to consider the scope of your project for each minor feature you incorporate into the system. I discovered that, in addition to effectiveness and performance, producing high-quality code also requires thoughtful consideration of the structure and design of the code. Working with an iterative model also gave me a deeper understanding of how software engineering ideas are applied in the ARDE's ongoing research and development of the country's military sector.