Vision-based Multi-robot Tracking of a Target with Aerial and Ground Robots

(Occlusion Avoidance & Deep Learning Tracking for Aerial Robots)

Vamshi Kodipaka

Laboratoire d'Analyse et d'Architecture des Systèmes
(LAAS - CNRS) Toulouse, France

Supervisors:
Prof. Antonio Franchi
Prof. Daniel Sidobre

A Thesis Submitted for the
Degree of MSc in Computer Vision (MSCV)
Université de Bourgogne, France

· 2020 ·

Abstract

Recent research in Aerial Robotics defines Visual Tracking Methods deployed on Quadrotors. The idea is to study the existing state-of-the-art real time tracking algorithms and free to adopt the existing ones considering the problem of occlusion avoidance. Occlusions are mostly unpredictable and still ongoing research in tracking scenarios. The inability to track the object's features when some obstacle is encountered in between the camera field of view and moving target are usually termed as 'Phenomenon of Occlusion'. The physical world obstacles causing these phenomenon are called 'Occlusions'. Basically, occlusions are two types. One is Spatio-temporal: No occlusion, partial occlusion & full occlusion. Other one is Time sequences based: Regular time occlusions and Irregular time occlusions.

Deploying a tracking algorithm considering the Occlusion Avoidance is a challenging task. This will include exploitation of the following parameters: geometric, photo-metric, motion, time, camera, feedbacks, control loops, real time stability, sensors, UAV dynamics etc. One has to build his own environment and the required dataset in indoor or outdoor arena allotted to him. This work states the novel & spanking new solution for Occlusion Aviodance through the Time of Occlusion concept to drive or navigate or control the UAV in the 3D environment. Then after deploy the tracking algorithm for tracking the moving target. Deep Learning Tools are observed as profound methods and provide effective results (enabling optimization efficiently).

After a rigorous study of 10 Standard Robotic papers and over 30 papers on state-of-the-art Deep Neural Networks, the latest evolutions like YOLO, RetinaNet and Joint Monocular 3D Tracking surely believed to be future milestones upon using these tracking networks on Aerial Vehicles. Each one of them is expected to do potentially well & therefore these algorithms are tested and reported with qualitative results.

Contents

List of Figures

Acknowledgments

No creation in this world is solo effort. Neither is this work. Patching some great works together became this thesis. I would like to thank all of them. Particularly, my sincere gratitude is with my supervisor Prof.Antonio Franchi (Head Researcher, RIS Team - LAAS CNRS and Professor in Mechatronics - University of Twente) for his constant guidance, motivation and support throughout the course of this work, which helped me to make successful contributions in solving the proposed challenges.

I am deeply indebted to my PhD colleague Mr.Martin Jacquet (with Prof.Antonio, LAAS CNRS) works on this ANR project MuRoPhen[1], who timely helped me to resolve the issues I face during the work. We together solved the problem of Time of Occlusion. He encouraged me through out, clarified doubts when I had questions/troubles in the internship and gave me many crucial suggestions to make sure I am in right direction.

I would like to thank my other supervisor Prof. Daniel Sidobre, Maître de conferences - University of Paul Sabatier-Toulouse, for the insights in basic Dynamics of UAV. Also I thank Prof. Simon Lacroix (Head of RIS Team, LAAS CNRS)

[1]EU MuRoPhen Project: https://www.murophen-project.eu/

providing me the work station. This helped me to setup the work environment at home and Dr. Matthieu Herrb for giving me the network setup configuration files. Thanks to these amazing people for taking care of me during this work and provided me freedom to choose implementation strategies and planning of work.

Thanks to Research Team (a group of 10 wonderful colleagues) with Prof.Antonio, for a good rapport & enabling delightful work environment to relieve the stress and stay active. Thanks to RIS Team[2] for orgainzing the Seminars time-to-time and keeping me updated. Thanks to Teams of Prof. Andrew Ng[3] and Prof. Vijay Kumar[4] from Coursera online.

Special thanks all my professors at University of Burgundy and Team VIBOT, for shaping me with the coursework and projects, to be ready for this research work. I would like to thank amazing friends who always stayed with me in the process of knowledge exchange and cultural exchange. Finally, thanks to my family for their great support during this COVID times and pandemic crisis.

[2]RIS Team: https://www.laas.fr/public/en/trombinoscope-ris
[3]Deep Learning.AI: https://www.coursera.org/specializations/deep-learning?
[4]Coursera Aerial Robotics: https://www.coursera.org/learn/robotics-flight?

Chapter 1

Introduction

Challenges are what make life interesting and overcoming them is what makes life meaningful

-Joshua J Marine

1.1 Background

Robust object tracking along with *occlusion handling* is a challenging problem in many applications of computer vision, such as motion based recognition [21] systems, automated surveillance, human computer interaction, video indexing, traffic monitoring, vehicle navigation etc. Capturing images with high quality and good size is becoming easier as technology evolves, with rapid improvements in quality of capturing device at a more and more accessible price. On the other hand, many algorithms and technologies have been developed to automate monitoring the object, and it is a very active field of research. In this field, target tracking [18] is used in various security and safety applications to detect various situations or phenomenon. The main objective is the detection of one or several objects of interest (often in motion), tracking of these objects in consecutive frames, and, later, the analysis of the objects motions to understand their behaviour. Over the last years, object detection and tracking have been introduced and the evolution is carried on in present times.

Object detection in videos involves verifying the presence of an object in image sequences and possibly locating it precisely for recognition. Object tracking is to monitor an *object's spatial and temporal changes during a video sequence*, including its presence, position, size, shape, etc. In short, it is a method in which a sparse set of features is often traced. *Motion detection* [10] is often observed from a static camera like in surveillance systems. *Object localization* focuses on a region of interest in the image. Often, only interest points found which are used later to solve the correspondence problem. But in motion segmentation, images are segmented into region corresponding to different moving objects.

In 2007, J.Pan and B.Hu [44] introduced a robust occlusion handling mechanism by developing adaptive progressive occlusion and variant mask matching techniques, solved occlusion problem by comparing motion characteristics between target and image blocks. This method has the advantage of explicitly detecting the outliers in a progressive manner & solves motion errors too but fails to note target speed. Its target speed relative to occlusion is low and other issue was occlusion & target appears same. X.Cheng and team [26] proposed an anti-occlusion observation model in 2016, which recovers multi-views automatically in video frames using homography. It is advantageous as there is no need for the 3D coordinate points. It is used as 3D Tracker but if once loses the tracks hard to recover. Number of cameras required is more in this method.

Later in 2017, F.Sardari and team [51] developed a hybrid occlusion free tracking using particle filter through a modified galaxy search meta-heuristic method. *Particle Filter, Spiral Chaotic Move, Local Search* forms higher level state transition model and appearance features. It is proven to be computationally complex, has memory issues along with many iterations and size changes, low illumination, partial or full occlusion, position variations are other issues noted. Z.Xun and team [65] launched an anti-occlusion correlation filtering tracking for quadrotors in 2018. It solves ridge regression problem in Fourier Domain (with 2D Gaussian) by Discriminative Correlation Filter. It is effective for partially occlusions & if size of the object changes. Drift correction is introduced for the stable tracking. It is advantageous when asked to extract a sample set of suppressing the background cluster, robustness and more tracking stability.

B.Penin and team [46] came up with a vision-based reactive planning while avoiding collisions and occlusions for tracking in the same year. It states spherical

projection of a target point in 3D (*beta*) - formulation and a differential flatness & mapping procedures where trajectory representation is given by control points & time generation by *Libschitz continuity*. Model Predictive Control(MPC) [9] works with visual constraints with features expected to be in the camera field of view. These strategies [15] with B-splines knot span for stability and convex hull property are given. More information about B-splines is found here [2]. Field of view limits, more occlusions, collision with obstacles are other drawbacks. Simulation video is found here[1]. ETH's Dronet [43] which used Udacity dataset and Collision dataset (2018) on a CNN generating a steering angle and collision probability, considered 3 cameras information from IMU, GPS, gear, brake, throttle. It has got mean squared error, binary cross entropy, epoch=10, learning rate=0.001 & decay=10^{-5}. This could achieve average accuracy with many layers and processing times (fps) but has real time applications both indoors and outdoors. The equipment used are Parrot Bebop 2.0 drone, visual odometry system, Intel i7 2.6GHz CPU that receive images at 30Hz(Wifi). For supplementary video refer link[2]. These significant contributions led foundation to use latest networks for the development of tracking schemes solving occlusion problems.

1.2 Statement and Motivation

The classical problem of occlusions are always featured in tracking schemes. The ultimate goal of technological advancements in vision is to improve the quality and speed of target detection and tracking systems. It has to face with all general problems in computer vision, like illumination changes, changes in the appearance of moving objects, presence of unpredicted motion, complex backgrounds, moving shadows, camera failures etc. In this context, moving objects detection in video streams is a promising yet challenging task for modern developers. Neural networks provide many possibilities to improve the accuracy of moving object detection, but require access to greater computational resources. This thesis gives a new look upon *Occlusion Avoidance in Target Tracking*. Implementing a real-world robotic system that applies the novel contributions of the research and disseminating the findings and advances associated with this project to the

[1]Vision-Based Collisions and Occlusions Avoidance: https://youtu.be/mvvF1I72HM8
[2]ETH Zurich's Dronet: https://youtu.be/ow7aw9H4BcA

scientific community.

1.3 About LAAS

The Laboratory of Analysis and Architecture of Systems (LAAS)[3] is a CNRS[4] research unit linked with the Institute for Engineering and Systems Sciences (INSIS)[5] and the Institute of Information Sciences and their interactions (INS2I)[6].

It is located at Toulouse, France and is associated with University of Toulouse, INSA and INPT. LAAS digs in Information Sciences and Technologies and address complex systems, and at different scales, theories, methodologies and tools for modeling, designing and controlling them. The lab has several collaborative projects international, national and regional industries of all scales. Various researches of the lab: integrated, distributed, embedded, mobile, autonomous and robotics, micro and nano, biological systems with applications in aeronautics and space, telecommunications, transports, production, services, security/defense, energy management, health-care & sustainable development. There are more than 450 researchers and PhD students, internship students, working in the lab supported by administrative staff. The lab's scientific research is distributed into 8 main departments, in which Robotics is one of it.

The Robotics department subdivided into three groups: 1)GEPETTO[7] - works on motion of anthropomorphic systems, 2)Robotics Action and Perception (RAP)[8] - works on perception, sensor-based motion and sensor integration in robotics, 3)Robotics and InteractionS (RIS)[9] - works on autonomous mobile machines that integrate perception, reasoning, learning, action and reaction capabilities. The RIS group works mainly on perception, reasoning, learning, action and reaction. This thesis is carried out in RIS (domain of learning & perception on heterogeneous multi-robot systems). We are collaborating with GEPETTO and RAP for the integration of pose estimation module on humanoid robot 'Pyrene'.

[3] https://www.laas.fr/public/en
[4] http://www.cnrs.fr/index.html.
[5] https://insis.cnrs.fr/
[6] https://ins2i.cnrs.fr/
[7] More information can be found at http://projects.laas.fr/gepetto/
[8] More information can be found at https://www.laas.fr/public/en/rap
[9] More information can be found at https://www.laas.fr/public/en/ris

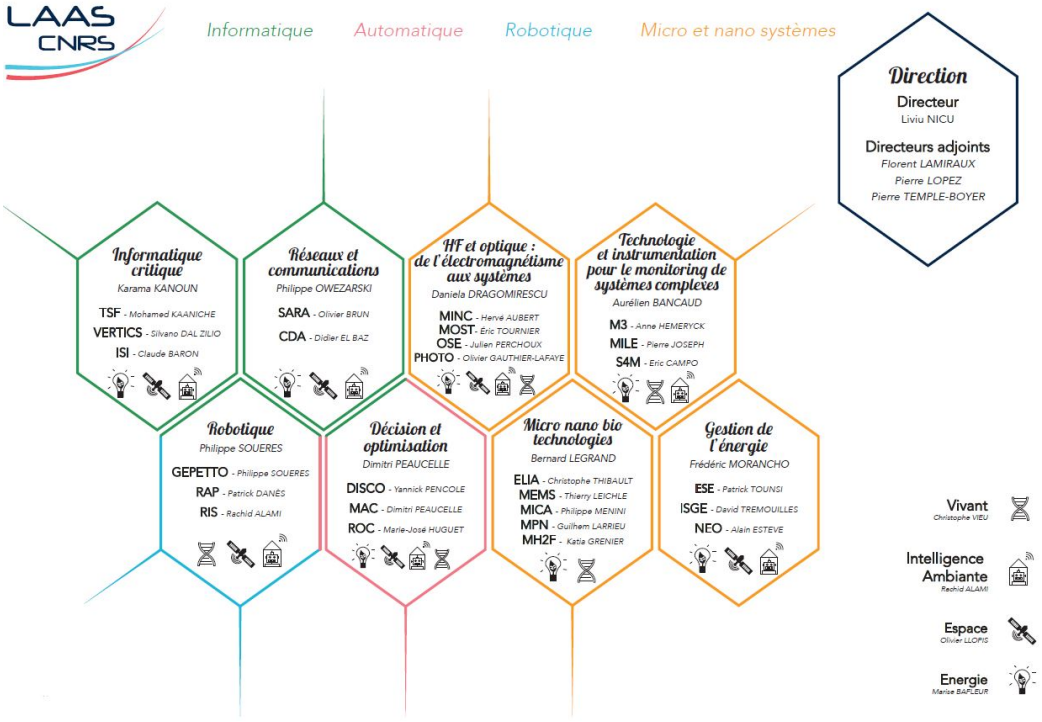

Figure 1.1: Work Place: LAAS on the whole

1.4 Project Objectives & Planning

In order to achieve the specified problem statement, the following project objectives are noted.

1. Literature Review on Tracking Schemes in Robotics - Summary
2. Develop a Mathematical Model for Occlusion Avoidance for targets with linear and quadratic motion
3. Seek for the knowledge in Deep Learning and Basics of Aerial Robotics
4. A Comparative study over the latest Object Detection schemes - Summary
5. Implement YOLO, RetinaNet. Report Joint Monocular 3D Depth Tracking

6. Estimate Depth based tracking & integrating with robotic platforms (if we have time - till Jul31)

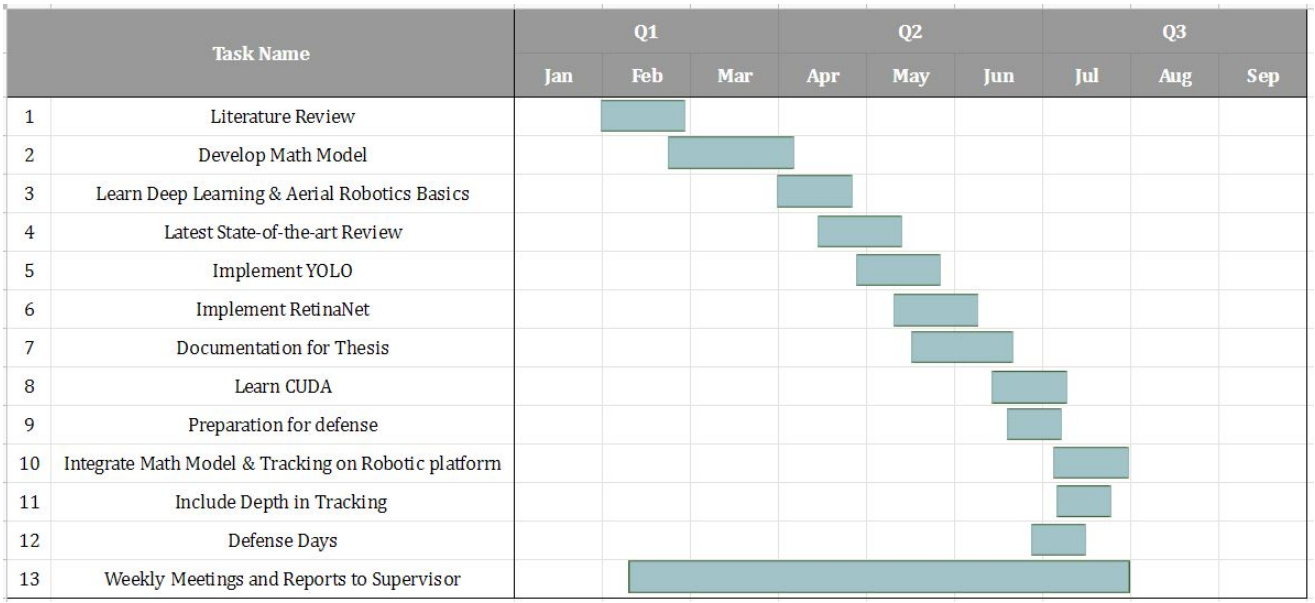

Figure 1.2: Gantt Chart of work

The work is managed and planned effectively as shown in Figure 1.2.

This thesis draft is organized as follows: Chapter 2 presents the methodology of the project. Chapter 3 introduces the Mathematical Modelling to achieve occlusion avoidance. Chapter 4 discusses over State-of-the-art Detection algorithms, summarizes the current trends and selected algorithms. Chapter 5 explains the implementation of YOLO and RetinaNet. Chapter 6 focuses on results and discussions. Chapter 6 gives the conclusion and future work of this internship.

Chapter 2

Methodology

Those who wish to succeed must ask the right preliminary questions

-Aristotle

In this chapter, we shall go through the methodology developed for the internship work. That is, we go through the basics of quadrotor dynamics for rigid body motion, proposed control loop & purpose of deep learning in this work.

2.1 Basics in Quadrotor Dynamics

Quadrotors are the class of Aerial Vehicles those which consists of four propellers mounted on a support frame used in navigating in/around the obstacles, in trajectory planning indoors/oudoors, sensing the Tags while navigating. Based on these operations and sizes there exists uavs, remotely piloted vehicles, aerial robots & Drones. Quadrotors with the help of propellers attains Roll, Pitch & Yaw[1] which inturn responsible for maneuverability and agility. In order to enable autonomous flight, there are a few key components that we need in every robot.

Number one, 'state estimation' which is the ability of the vehicle to estimate its position, its orientation, and velocity. (also defined as rate of change of position and orientation). Two, 'control' which is the vehicle ability to compute its control commands. Based on where it needs to go and based on what its estimate of its

[1]Euler angles: https://en.wikipedia.org/wiki/Euler_angles

current state is, the vehicle must be able to compute the commands that need to be sent to the motors or the rotors, and have them rotate at the appropriate speeds to achieve the desired action. Three, 'mapping' which has some basic capability to map its environment. If it does not know what the surrounding environment looks like, then it's incapable of reasoning about this environment and planning safe trajectories in this environment. Four, 'planning' which is defined as given a set of obstacles and given a destination, the vehicle must be able to compute a trajectory, a safe path to go from one point to another.

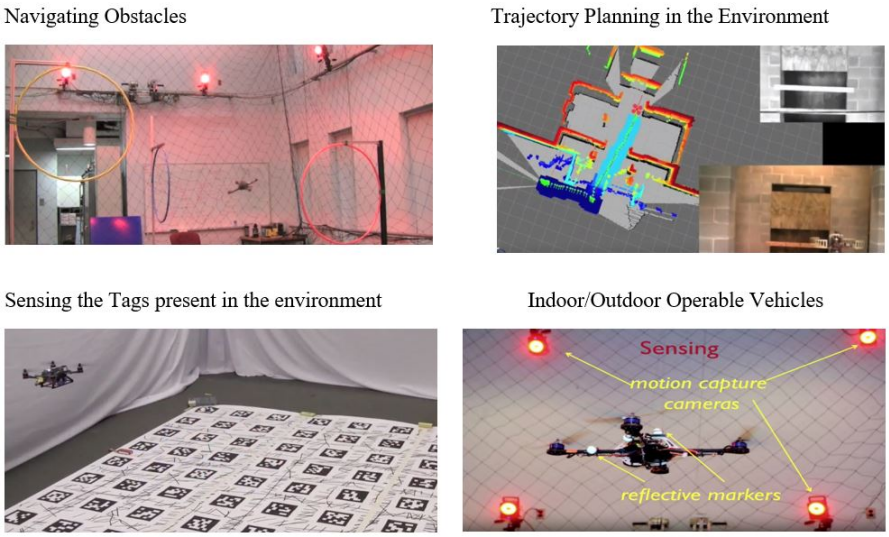

Figure 2.1: UAV in expert test environments

One question is how do you get the robot to steer or to yaw? So if you want to move the vehicle, just translating it along the horizontal direction, one have to pitch the robot forward so that the trust factor points in the horizontal direction. That allows the vehicle to accelerate forward. But then when you get close to the destination, you want to stop the vehicle. One have to pitch it in the opposite direction, creating a reverse thrust that allows it slow down when it gets to its destination. And then you have to pitch it back to equilibrium. And this way we will be able to operate quadrotors both indoors and outdoors. There's no GPS and because these vehicles are small and maneuverable, they can find themselves in settings where it's very difficult to communicate directly with the vehicle. We also want these vehicles to move very quickly and maneuver through these complex environments. How do we navigate without GPS, without

external motion capture cameras or any other kinds of external sensors? The quick answer is SLAM. Here to state some applications of quaarotors: Agriculture, Construction, Archaeology, Photography, Tracking & Swarm drones for rescue.

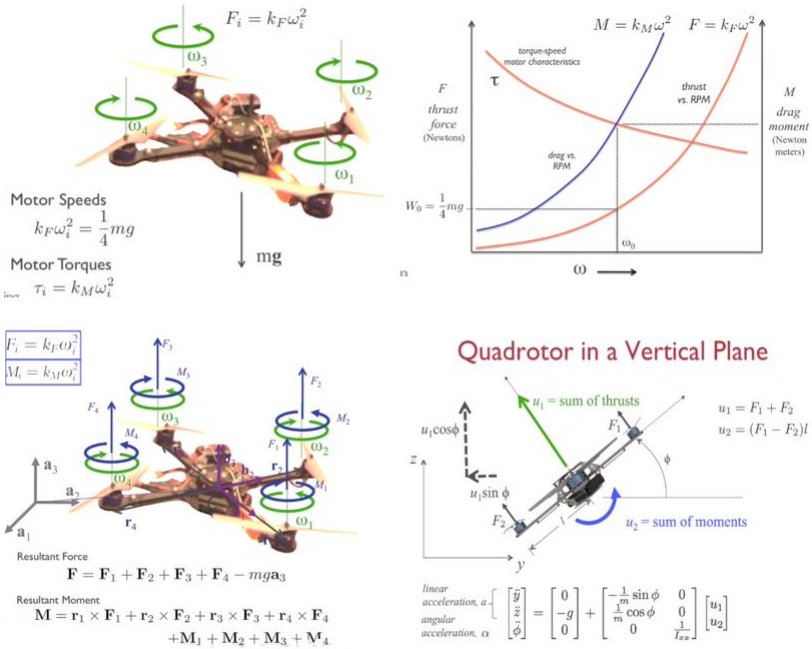

Figure 2.2: Hover and Smooth Flight

Component Selection varies with Frames, Motors, Propellers, Autopilot means electronic board, price, weight, sensors (3D), processor, IMU[2], memory USB ports, SSD card slot, Power DC and also Processing and Communication slot: UART + PWM along with on board sensors like Laser scanners, Cameras. Power requirement vs Thrust vs Power Consumption vs Energy vs Sensors: Quadrotors are ten times more inefficient, than possibly the most inefficient man on Earth, as it runs the hundred meters race in ten seconds. Robots are inefficient, actually hovering is an inefficient mechanism, so we needs lots of power to power our robots. And if you look at lithium polymer batteries which represent the best choice of batteries today, they don't carry a lot of energy. So we try to build smaller and lighter quadrotors. When thinking about the payload we want to think about the power consumed in addition to the thrust to weight ratio. Agile Robots are those can start from a position of rest, accelerate pretty quickly to

[2]IMU: https://en.wikipedia.org/wiki/Inertial_measurement_unit

a maximum speed, stop when it sees obstacles, and then accelerate again to a maximum speed.

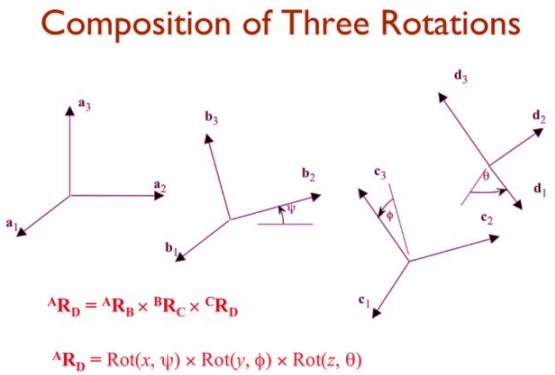

Figure 2.3: Three Rotations with the Euler angles in the free space

Concept of a reference frame: We associate with each position and orientation a reference frame. A rigid body displacement is nothing but a map from this collection of points in the object, to its physical manifestation in R3, rigid body transformations, or rigid body displacements, satisfy two important properties. They preserve lengths and they preserve cross products. This proves that for a rigid body displacement, orthogonal vectors are mapped to orthogonal vectors, and further, g* preserves inner products. For more information refer [6].

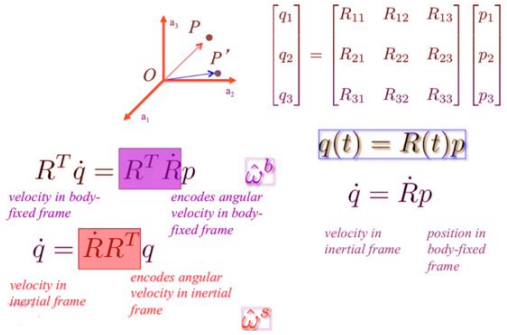

Figure 2.4: Derived Rotation Matrix and Angular Velocity

That is, if you compute the inner product of two vectors before displacements and in a product of the same two vectors after the displacement, those inner products are the same. Transformations generally referred to relationships between

reference frames attached to two different rigid bodies, while displacements gives relationship between positions & orientations of a frame attached to the rigid body as it moves around. Observe mutually orthogonal unit vectors attached to every rigid body, if it's a transformation, we're referring to two different rigid bodies. If it's a displacement, it is two distinct positions & orientations of the same rigid body. We can write the mutually orthogonal unit vectors in one frame as a linear combination of the mutually orthogonal unit vectors in other frame.

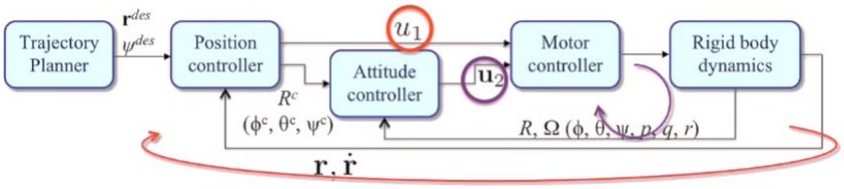

Figure 2.5: 2D/3D Quadrotor control

Properties of Rotation Matrix: Orthogonality, det of R is 1, product of Rx and Ry is Rotation, inv(R) is Rot. Rotation about x,y,z with Roll, Pitch, Yaw. Eg: Earth xyz axis. Remember, Euler quoted that any rotation can be described by three successive rotations about linearly independent axes. R_{3x3} is not one to one considering 3 Euler angles XYX, ZYZ - Euler angles[3] (Body Fixed Axes). Are the three axes linearly independent? The two z-axes are not co-linear, they are independent. If this condition is satisfied, then the three angles are Euler angles, and they can parameterize the set of rotations. And one of the theta =0 analogous to earth's poles. Rotation on one axis vs angular velocity: ang.velocity in the axis we can then characterize the rigid-body displacement G using the rotation matrix R refer figure 2.4 As per Euler's principle displacement of a rigid body, we derived at the eqns stated in above figure 2.4. And as per Newton-Euler Equations[4], considering moments and inertia, we derive *dynamics equations of motions (for a quadotor)*. Then we have 2D/3D position control loop refer figure 2.5.

Attitude control is the process of controlling the orientation of an aerospace vehicle with respect to an inertial frame of reference or another entity such as the celestial sphere, certain fields, and nearby objects, etc. Time, motion and trajectories: orientations, input, order of system (ODE). Hover Configuration involves

[3] https://www.coursera.org/lecture/robotics-flight/formulation-IxGuA
[4] https://www.coursera.org/lecture/robotics-flight/quadrotor-equations-of-motion-RRaaT

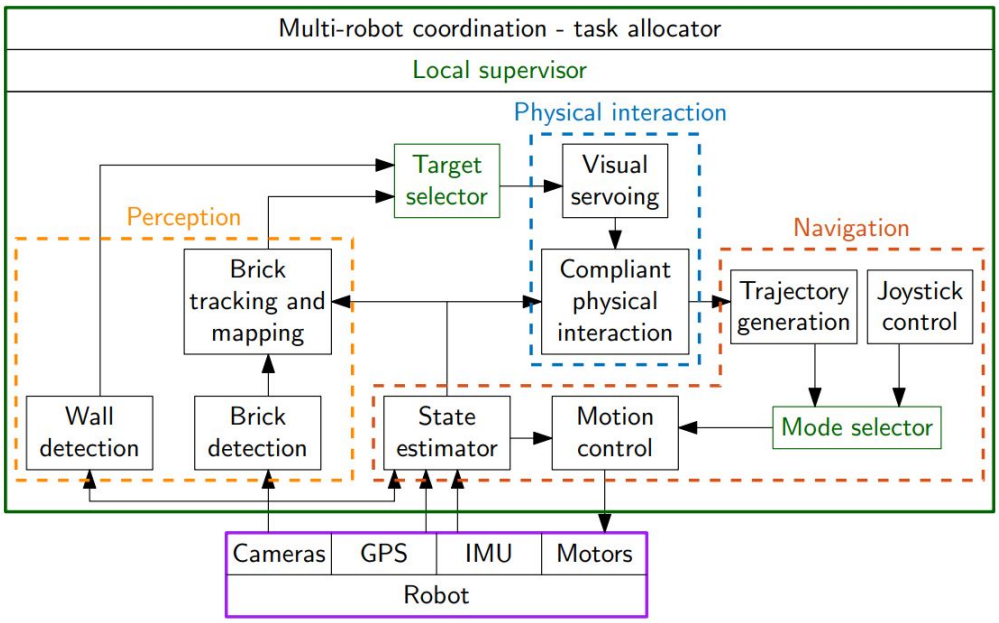

Figure 2.6: Sample Estimation & Control Architecture

Motion Planning for a quadrotor that is obstacle avoidance, min jerk trajectory, waypoint navigation, min velocity trajectory, linearization quadrotor modelling eqns, sensing and optimization: SLAM & structure from motion. Finally, to overcome Euler angle problems, we go for quaternions and UAVs tends to has non-linear control and stability with basin of attraction & differential flatness.

2.2 Proposed Control Loop

In general, as per whole project demands to develop similar to this Sample Architecture[5] as in Figure 2.6. Also refer other Sample Architecture2 [5] here. And my internship focuses on development of Mathematical Model for the Occlusion Avoidance and Tracking Algorithms those can be deployed on the Aerial Robots. Therefore, my task as for as internship is concern is simplified to the Figure 2.7.

Imagine a quadrotor is expected to be mounted or has a flight as shown in Figure 3.2 Briefly, the control loop of my work is given in four blocks stages.

1. To collect the data or create a custom dataset images from the camera.

[5]Courtesy: LAAS Scheme for MBZIRC International Robotics Challenge 2020, Abu Dhabi

2. Use a tracker to find the location or detect of the target and occlusion.

3. Given the state of the robot, the position estimate of target and occlusion compute the time of occlusion using the formula given t_{occ} (either case1 or case2 as per the trajectory of the target; explained in chapter3).

4. The computed t_{occ} is then fed to the controller to adjust the position of the quadrotor to avoid occlusions in the real world.

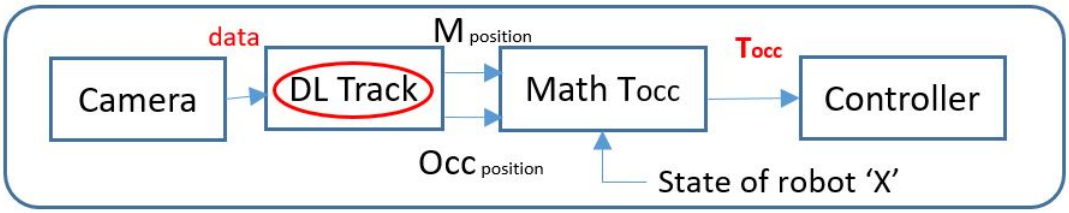

Figure 2.7: Proposed Control Architecture

2.3 Why Deep Neural Networks?

Deep Learning allows smarter way of filtering the pixels in an image with smarter computation cost performing tasks like recognition, classification, classification with localization, object detection, object tracking etc. by giving high scope for fine tuning (that is for accuracy and precision). Transfer learning is an extraordinary feature of Deep Neural Networks.

Classification with Object Localization we can get object recognized, not only that but also we get the bounding box for the object in an image. In this way we get the ground truth box. Sequential sliding search through the defined windows and passing them through the *Convolutional Neural Networks*[6] is the very basic approach. Convolutional implementation of these sliding windows are more computationally efficient than the previous one as (you can implement the entire image, make all the predictions at the same time by one forward pass through this big convent and have it recognize the position of the object, unlike the use of Fully Convolutions layers with softmax). But again it still has a problem of

[6]CNN: https://en.wikipedia.org/wiki/Convolutional_neural_network

not quite outputting the most accurate bounding boxes. This approach of sliding window with a CNN gives maybe the boxes what is the best match but does not really match up boxes perfectly with the position of the object.

Many examples, we actually have a slightly wider rectangle or slightly horizontal aspect ratio or maybe slightly longer rectangle or slightly vertical aspect ratio. Is there a way to get this algorithm to outputs more accurate bounding boxes? A good way to get this output more accurate bounding boxes is with the YOLO algorithm. YOLO stands for, You Only Look Once. Ofcourse, other than YOLO, standard tracking algorithms like RetinaNet (from Facebook AI Research) and 3D Monocular Depth Tracking Algorithms are expected to be output more accurate bounding & considered as the mile stones in future research.

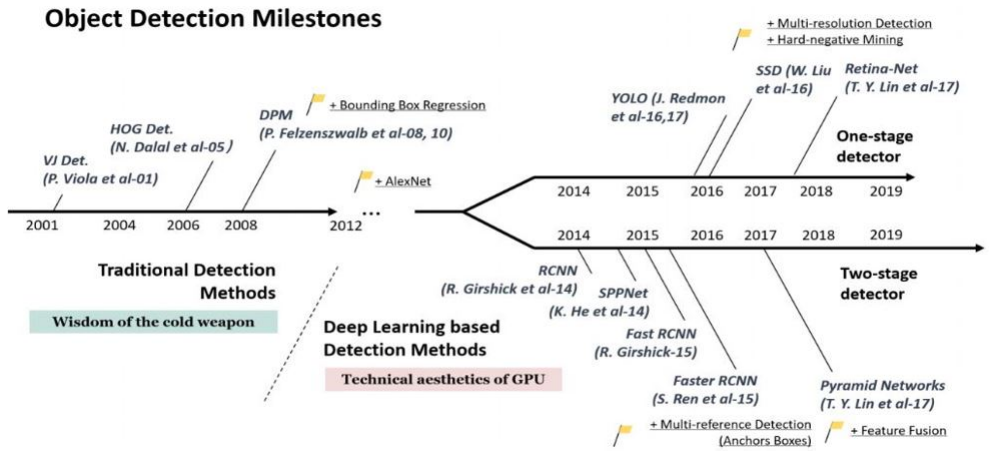

Figure 2.8: Object Detection Paradigm

So we have to know the evolution of DNN mainly interms of Two-stage vs One-stage Detectors. Models in the R-CNN family are all region-based. The detection happens in two types:

1. If the model proposes a set of regions of interests by select search or regional proposal network and the proposed regions are sparse as the potential bounding box candidates can be infinite those are refereed as 'Two stage Detectors'. Then a classifier only processes the region candidates. They are slow compared to one stage Detectors.

2. The other different approach is to skip the region proposal stage and runs detection directly over a dense sampling of possible locations. This is how

a one-stage object detection algorithm works. This is faster and simpler, but might potentially drag down the performance (accuracy) a bit.

Trending common model architectures used for object detection are given as: R-CNN Family, SSD (Single Shot MultiBox Defender), YOLO (You Only Look Once), Objects as Points, Data Augmentation Strategies ans so on.

2.4 Conclusion

Fine tuning the network parameters is easy with Deep Learning Networks (Train and Test approach) which makes our system more prone to accuracy in terms of mAP (mean Average Precision) and expected to outperform the classical vision trackers. One should also have to consider the speed as a performance of the tracking system. To go beyond the CPU clock, *Graphical Processing Units* (GPU) helps in parallel computing. A GPU[7] is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device. Thus this hardware helps tracking system to be more robust interms of efficiency.

[7] https://en.wikipedia.org/wiki/Graphics_processing_unit

Chapter 3

Mathematical Model for Occlusion Avoidance

'One picture is worth more than ten thousand words'

-anonymous

In earlier chapters, we came across the methodology for work. This chapter exclusively emphasis on the novel time of occlusion concept proposed as part of the internship work, what is occlusion and importance of avoiding occlusions. Then, in short, distinguish two time of occlusion formulas for two distinct considered cases. Our math computation moves around this model figure 3.2. Clearly, time of occlusion is derived from rigid body motion (rotations and translations with respect to 3D frame of reference) and is explained for straight line trajectory & quadratic motion variation of moving target.

3.1 t_{Occ} - Diagram & Computation

The inability to track the object's features when some obstacle is encountered in between the camera field of view and moving target are usually termed as *'Phenomenon of Occlusion'*. The physical world obstacles causing this phenomenon are called 'Occlusions'. In spatio-temporal context usually it is referred to be no occlusion, partial occlusion & full occlusion. Self-occlusion most frequently arises

while tracking articulated objects when one part of the object occludes another. In time sequences context they are described as regular time occlusions and irregular time occlusions. Inter-object occlusion occurs when two objects being tracked occlude each other whereas occlusion by the background occurs when a structure in the background occludes the tracked objects. For clear understanding of our case scenario, refer figure 3.1

Let us consider a freely flying quadrotor with a downward looking camera (with a fixed camera field of view) mounted to the frame & has a fixed occlusion position in the 3D world as shown in figure 3.1. Initially, consider the target M is at rest at *a1*, then a quadrotor at position **C** can easily keep track of the target. Now, assume the target attains velocity, currrently at position *b1* and moves towards occlusion, to keep track of the target we feed the control input to move our quadrotor to position **B**. Till now we don't have any problem in keeping track of the target.

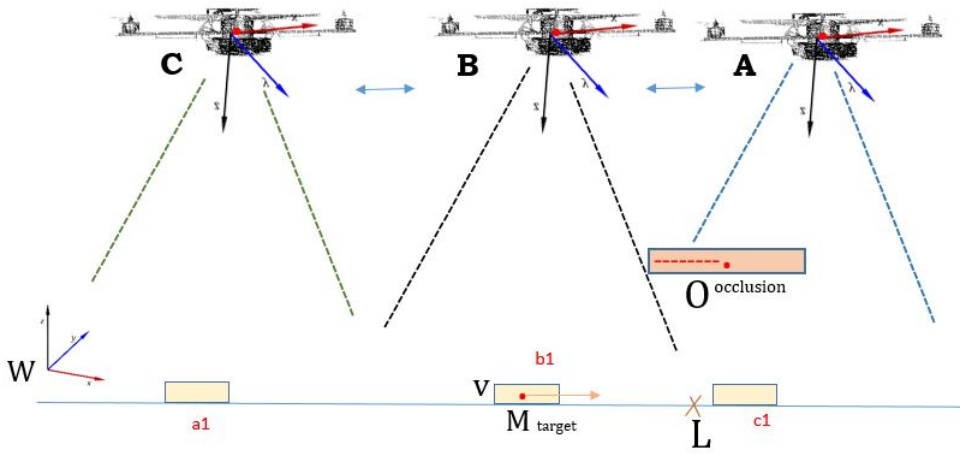

Figure 3.1: Concept of Time of occlusion describing scenarios

Now, once the quadrotor start moving to right to the point 'L', the quadrotor at position B loses its hold on tracking the target M due to the obstruction of occlusion object. Now we should feed our controller (refer control loop 2.7) a input to move our quadrotor to other virtual position B1 or A in such a way that our quadrotor attains tracks again. Therefore, the instinct 'L' can be called as *Shadow of occlusion*, a line at which the quadrotor loses the tracks of the target. So, here when the target is in position *b1* time of occlusion metric comes handy.

Moving target is allowed to track using the camera input information from camera mounted on the quadrotor. Then while quadrotor is at position **B**, given state of the quadrotor, the position of target & occlusion with the help of tracking bounding boxes (*depth included,* measured relative to the 3D world coordinates) are used to compute the time instant at which object is going to be occluded as described in the below sections. So, this metric of time instant at which object is going to be occluded is termed to be as *Time of Occlusion*.

Now this metric of time can be effective interms of relative occlusion time instances. This time instances inturn helps to calculate the time difference of current instance of target to the time instance at the point of occlusion. This is termed to be *'Time to Occlusion'*. This differential time help the control to take a decision to move in a proper direction to avoid occlusion.

3.2 Case 1: For straight line trajectory

Let's define the scene at a given instant t. Refer Figure 3.2. M is the moving object, L is the shadow of line of occlusion, O is the center of occlusion body, C is the center of the UAV frame and W is the world frame. Following Definition, Assumptions and Problem defines this case.

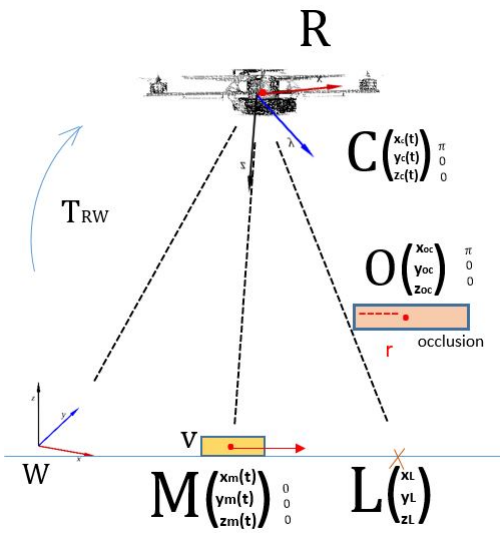

Figure 3.2: Working Model for Time of Occlusion

3.3 Definitions

Let ${}^{W}C = \begin{bmatrix} x_C & y_C & z_C \end{bmatrix}^{\top}_{W}$, ${}^{W}\eta_C = \begin{bmatrix} \phi_C & \theta_C & \psi_C \end{bmatrix}^{\top}_{W}$ and ${}^{W}R_C$ the position, orientation (euler angles) and associated rotation matrix of the camera in world frame, denoted $\mathcal{F}_W$ respectively. Similarly, we define ${}^{C}M$ and ${}^{C}Occ$ as positions in $\mathcal{F}_C$ of marker (also called target) and occlusion obstacle. The orientation of the marker and occlusion are considered aligned with the frame axis of $\mathcal{F}_C$, hence ${}^{C}\eta_M = {}^{C}\eta_{Occ} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^{\top}_{C}$. The target considered ponctual (no size) is on the constrained on the ground, hence ${}^{W}z_M = 0$. Refer Figure 3.2.

The occlusion obstacle is considered planar (e.g. a table), with $0 < {}^{W}z_{Occ} < {}^{W}z_C$. We call $\overline{Occ}$ the border of the occlusion obstacle, which is the limit of the camera occlusion. The distance between Occ and $\overline{Occ}$ is r considering the occlusion body to be a circular disk like a table top. This is also called as *Radius of Occlusion*. The point L corresponds to the intersection of the line $(C, \overline{Occ})$ and the ground. Now, we shall dive into case1, please refer figure 3.3

3.4 Assumptions

We assume that the target M is always in the field of view of the camera C.

We consider the occlusion obstacle fixed in $\mathcal{F}_W$.
For the time being, we also assume the camera (C, η_C) fixed in $\mathcal{F}_W$.
The target, however, is moving at constant speed $v_m = \dot{M}$.
Note that this assumption makes M a function of the time. The position of the target at an instant t is then denoted $M(t)$.

Finally, we define t_{Occ} as the time needed for the target to be occluded from the camera sight. It is the time needed by the target M to reach the position L at constant speed v_M. So:

$$t_{Occ} = \frac{\|M - L\|}{v_M} \tag{3.1}$$

3.4.1 Problem

In order to compute t_{Occ} from the drone perspective, we have to use what is known by the robot. We assume known the following:

$$^{W}C, ^{W}\eta_{C} \tag{3.2a}$$

$$^{C}M, ^{C}\dot{M} = ^{C}v_{M}, ^{C}\ddot{M} = 0 \tag{3.2b}$$

$$^{C}\overline{Occ} = \begin{bmatrix} x_{Occ} - r \\ y_{Occ} - r \\ z_{Occ} \end{bmatrix}_{C} \tag{3.2c}$$

We want to express t_{Occ} as a function of these variables.

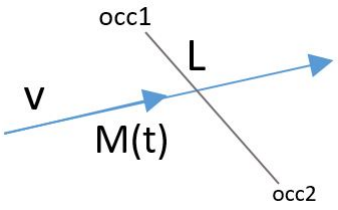

Figure 3.3: Linear Motion for Occlusion Interpretation

3.4.2 Solution

Let's start by expressing L in camera frame $\mathcal{F}_W$.
Since it is the projection on the ground of the line $(C, \overline{Occ})$, we have

$$^{W}z_{L} = 0 \tag{3.3a}$$

$$^{C}L = \lambda \times ^{C}\overline{Occ} \tag{3.3b}$$

$$^{W}L = ^{W}t_{C} + ^{W}R_{C}{}^{C}L \tag{3.3c}$$

$$= ^{W}C + ^{W}R_{C}{}^{C}L \tag{3.3d}$$

$$= ^{W}C + \lambda ^{W}R_{C}{}^{C}\overline{Occ} \tag{3.3e}$$

So, coupling eq. 3.3a with the eq. 3.3e, we have this equation defining λ:

$$0 = {}^{W}z_C + \lambda \begin{bmatrix} -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \begin{bmatrix} x_{Occ} - r \\ y_{Occ} - r \\ z_{Occ} \end{bmatrix}_C \tag{3.4a}$$

$$0 = {}^{W}z_C + \lambda \left[-s\theta(x_{Occ} - r) + s\phi c\theta(y_{Occ} - r) + c\phi c\theta z_{Occ} \right] \tag{3.4b}$$

Now that we know λ, we also know the coordinates of L in camera frame from eq. 3.4b.

$$t_{Occ} = \frac{\|M - \lambda \overline{Occ}\|}{v_M} \tag{3.5}$$

where $\lambda = \frac{{}^{W}z_C}{s\theta(x_{Occ}-r) - s\phi c\theta(y_{Occ}-r) - c\phi c\theta z_{Occ}}$

This gives the formula for t_{occ} for case1.

3.5 Case2: Moving target with time-varying speed

We consider the same problem as before, but now assuming that the target moves with constant acceleration instead of constant speed.

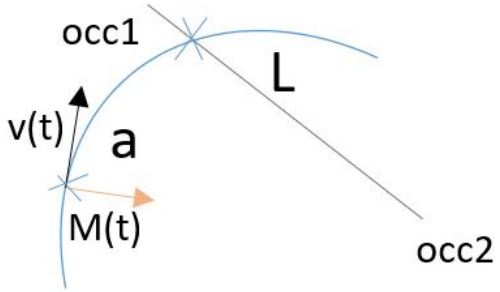

Figure 3.4: Quadratic Motion for Occlusion Interpretation

From camera measurements, we have:

$${}^{C}M(t), {}^{C}\dot{M}(t) = {}^{C}v(t), {}^{C}\ddot{M} = a \tag{3.6}$$

First, compute the projection w.r.t. to camera ("shadow") of $\overline{O}cc_1$ and $\overline{O}cc_2$. Let's call $(d) : x + ay + b = 0$ the line passing by these two points.

The position equation of $M(t)$ is:

$$M(t) = M(t_0) + v(t - t_0) + \frac{1}{2}(t - t0)^2 a \tag{3.7}$$

Call $L = \begin{bmatrix} x_L \\ y_L \\ z_L = 0 \end{bmatrix}$ the intersection between (d) & the curved path of target M. Here, t_{Occ} is the duration between current instant t and the instant at which the target will be in position L: $L = M(t_{Occ})$. So we have the following relations:

$$0 = x_L + ay_L + b \tag{3.8a}$$

$$x_L = x(t) + t_{Occ}v_x(t) + \frac{1}{2}t_{Occ}^2 a_x \tag{3.8b}$$

$$y_L = y(t) + t_{Occ}v_y(t) + \frac{1}{2}t_{Occ}^2 a_y \tag{3.8c}$$

3.5.1 Solution

Substituting 3.8b and 3.8c in 3.8a, therefore 3.8a becomes:

$$0 = [x(t) + t_{Occ}v_x(t) + \frac{1}{2}t_{Occ}^2 a_x] + a[y(t) + t_{Occ}v_y(t) + \frac{1}{2}t_{Occ}^2 a_y] + b \tag{3.9a}$$

$$0 = [x(t) + ay(t) + b] + t_{Occ}[v_x(t) + av_y(t)] + t_{Occ}^2[(a_x + aa_y)/2] \tag{3.9b}$$

solving quadratic equation for roots:

$$t_{occ} = \frac{-[v_x(t) + av_y(t)] + sqrt([v_x(t) + av_y(t)]^2 - 2[(a_x + aa_y)][x(t) + ay(t) + b])}{(a_x + aa_y)} \tag{3.10a}$$

$$t_{occ} = \frac{-[v_x(t) + av_y(t)] - sqrt([v_x(t) + av_y(t)]^2 - 2[(a_x + aa_y)][x(t) + ay(t) + b])}{(a_x + aa_y)} \tag{3.10b}$$

Lets express occlusion line eqn first:

$$[y - y_{occ1}] = \frac{[y_{occ2} - y_{occ1}]}{[x_{occ2} - x_{occ1}]} \times [x - x_{occ1}] \tag{3.11}$$

so,

$$x - y[\frac{x_{occ1} - x_{occ2}}{y_{occ2} - y_{occ1}}] + \frac{[x_{occ2}y_{occ1} - x_{occ1}y_{occ2}]}{y_{occ2} - y_{occ1}} = 0 \tag{3.12}$$

occ line eqn3.11 is in the form $x_{occ} + uy_{occ} + v = 0$; say the slope is m_2
occ line eqn3.8a is in the form $x_L + ay_L + b = 0$; say the slope is m_1

Since they both are parallel lines as line (d) is projection of line of occlusion; their slopes are equal

then $m1 = m2$; so : $a1/a2 = b1/b2$ from these two lines
thereby we get a in 10a, that is : $\boxed{a = [x_{occ2} - x_{occ1}]/[y_{occ2} - y_{occ1}]}$

They have different 'b'. Now, how to compute b?

3.5.2 Finding b parameter

Let L1 (x_{L1}, y_{L1}) is projection of occ_1 point on to the ground. Let L2 (x_{L2}, y_{L2}) is projection of occ_2 point on to the ground .
Like in previous case; **for Occlusion Point:** ${}^{x}L1$:

$${}^{W}L1 = {}^{W}t_C + {}^{W}R_C{}^{C}L1 \tag{3.13a}$$

Splitting for $({}^{x}L1, {}^{y}L1, {}^{z}L1)$, then

$$x_{L1} = {}^{W}x_C + \lambda_1 \Big[c\theta c\psi(x_{Occ} - r) + (-c\phi s\psi + s\phi c\psi s\theta)(y_{Occ1} - r) + (s\phi s\psi + c\phi s\theta c\psi)z_{Occ1} \tag{3.14a}$$

$$y_{L1} = {}^{W}y_C + \lambda \Big[c\theta s\psi(x_{Occ1} - r) + (c\phi c\psi + s\phi s\psi s\theta)(y_{Occ1} - r) + (s\phi c\phi)z_{Occ1}\Big] \tag{3.14b}$$

$$0 = z_{L1} = {}^{W}z_C + \lambda \Big[-s\theta(x_{Occ1} - r) + (s\phi c\theta)(y_{Occ1} - r) + (c\phi c\theta)z_{Occ1}\Big] \tag{3.14c}$$

From 3.14a, 3.14b and 3.14c ; we rewrite the eqns; assuming

$\beta_1 = \Big[c\theta c\psi(x_{Occ} - r) + (-c\phi s\psi + s\phi c\psi s\theta)(y_{Occ1} - r) + (s\phi s\psi + c\phi s\theta c\psi)z_{Occ1}\Big]$

$\beta_2 = \Big[c\theta s\psi(x_{Occ1} - r) + (c\phi c\psi + s\phi s\psi s\theta)(y_{Occ1} - r) + (s\phi c\phi)z_{Occ1}\Big]$

$\alpha = \Big[-s\theta(x_{Occ1} - r) + (s\phi c\theta)(y_{Occ1} - r) + (c\phi c\theta)z_{Occ1}\Big]$

$$x_{L1} = {}^{W}x_C + \lambda_1 \Big[\beta_1\Big] \tag{3.15a}$$

$$y_{L1} = {}^{W}y_C + \lambda_1 \Big[\beta_2\Big] \tag{3.15b}$$

$$0 = z_{L1} = {}^{W}z_C + \lambda_1 \Big[\alpha_1\Big] \tag{3.15c}$$

From 3.15c we get λ_1, substituting it in 3.15b and 3.15c ; we get

$$x_{L1} = {}^{W}x_C - {}^{W}z_C * (\beta_1/\alpha) \tag{3.16a}$$

$$y_{L1} = {}^{W}x_C - {}^{W}z_C * (\beta_2)/\alpha) \tag{3.16b}$$

Here α, β_1 and β_2 are interms of occ_1 parameters

for Occlusion Point: ${}^{x}L2$:

$${}^{W}L2 = {}^{W}t_C + {}^{W}R_C{}^{C}L2 \tag{3.17a}$$

following same procedure as above ; we get

$$x_{L2} = {}^{W}x_C - {}^{W}z_C * (\beta_1^*/\alpha^*) \tag{3.18a}$$

$$y_{L2} = {}^{W}x_C - {}^{W}z_C * (\beta_2^*/\alpha^*) \tag{3.18b}$$

Here α^*, β_1^* and β_2^* are interms of occ_2 parameters.

Now we have occ_1 and occ_2; $(x_1, y_1), (x_2, y_2)$ using these points we write the eqn. of occlusion shadow line

$$[y - y_{occ1}] = \frac{[y_{occ2} - y_{occ1}]}{[x_{occ2} - x_{occ1}]} \times [x - x_{occ1}] \tag{3.18c}$$

Substituting 3.16a, 3.16b and 3.18a and 3.18b and solving 3.18c; we get:

$$x - y[\frac{\gamma_1 - \gamma_1^*}{\gamma_2 - \gamma_2^*}] + [\frac{{}^w yc(\gamma_1 - \gamma_1^*) + {}^w xc(\gamma_2^* - \gamma_2) + {}^w zc(\gamma_1^*\gamma_2 - \gamma_1\gamma_2^*)}{(\gamma_2 - \gamma_2^*)}] = 0 \quad (3.18d)$$

and as intermediate variables we get: $\gamma_1, \gamma_2, \gamma_1^*, \gamma_2^*$

$$\gamma_1 = \beta_1/\alpha, \gamma_2 = \beta_2/\alpha \quad (3.18e)$$

$$\gamma_1^* = \beta_1^*/\alpha^*, \gamma_2^* = \beta_2^*/\alpha^* \quad (3.18f)$$

eqn.3.18d resembles $x + ay + b = 0$ and gives eqn 3.8a; comparing them we get:

$$a = (\gamma_1 - \gamma_1^*)/(\gamma_2 - \gamma_2^*) \quad (3.18g)$$

$$b = [\frac{{}^w yc(\gamma_1 - \gamma_1^*) + {}^w xc(\gamma_2^* - \gamma_2) + {}^w zc(\gamma_1^*\gamma_2 - \gamma_1\gamma_2^*)}{(\gamma_2 - \gamma_2^*)}] \quad (3.18h)$$

substituting eqn 3.18e and 3.18f in 3.18g; we have:

$$a = [\frac{\beta_1 - \beta_1^*}{\beta_2 - \beta_2^*}] \quad (3.18i)$$

subtracting $(\beta_1 - \beta_1^*) and (\beta_2 - \beta_2^*)$and rewriting for a:

$$a = [\frac{(c\theta c\psi)(x_{occ1} - x_{occ2}) + (-c\phi s\psi + s\theta c\psi)(y_{occ1} - y_{occ2}) + (s\phi s\psi + c\phi s\theta c\phi)(z_{occ1} - z_{occ2})}{(c\theta s\psi)(x_{occ1} - x_{occ2}) + (c\phi c\psi + s\theta s\psi s\phi)(y_{occ1} - y_{occ2}) + (s\phi c\phi)(z_{occ1} - z_{occ2})}] \quad (3.18j)$$

now also substitute the orientation of the camera and solving 'a' becomes:

$$\boxed{a = [\frac{x_{occ1} - x_{occ2}}{y_{occ2} - y_{occ1}}]} \quad (3.18k)$$

'a' is same value as the one I obtained already (verified). Now substituting eqn 3.18e and eqn 3.18f in 3.18h, we get:

$$b = [\frac{{}^w yc(\beta_1 - \beta_1^*) + {}^w xc(\beta_2^* - \beta_2) + {}^w zc(\beta_1^*\beta_2 - \beta_1\beta_2^*)}{(\beta_2 - \beta_2^*)}] \quad (3.18l)$$

simplifying this gives the value of 'b',

$$\boxed{b = [\frac{{}^{w}xc(y_{occ2} - y_{occ1}) + {}^{w}yc(x_{occ1} - x_{occ2}) + {}^{w}zc(x_{occ2}y_{occ2} - x_{occ2}y_{occ1})}{(y_{occ2} - y_{occ1})}]} \tag{3.18m}$$

Thus eqn 3.18b represents the 'b' parameter we required.

Substitute values of **a** and **b** in eqn 3.10a; **you get formula of t_{occ} for case2**

3.6 Conclusion

Considering the definition, assumptions, problem statement and solutions for both cases, that is case1 and case2, we obtain the t_{occ}. Now this time of occlusion value is computed and feed to the controller as described in the figure 2.7. As per the derived formulae, it can be noted that Time of Occlusion in both cases depends on the target motion parameters and occlusion parameters. Thus, time of occlusion formula is derived for linear motion in case1 and time of occlusion for quadratic motion in case2 are explained clearly in this chapter. This work is considered the motivation for delpoyment of Deep Learning based tracking.

Chapter 4

State of the Art

'Work hard in silence, let success make the noise '

-anonymous

In process of Deep Learning achieving expertise from neural networks, to tuning hyperparameters, regularization and optimization, then to convolutional networks we observe so called Pre-trained Network Model. *Pre-trained network* is a network that has already learned to extract powerful and informative features from natural images and use it as a starting point to learn a new task. The majority of the pretrained networks are trained on a subset of the established database which are used in the open challenges. These networks are usually trained on more than a million images and can classify images into huge number of object categories, such as persons, keyboard, coffee mug, pencil, and many animals. Using a pretrained network with transfer learning is typically much faster and easier than training a network from scratch.

Transfer learning[1] is a research method that focuses on storing knowledge gained while solving one problem and applying it to a different but similar related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks. Here, knowledge means pre-trained weights. Let's see some establishments in DNN:

1. CNN Nets: AlexNet [35], VGG [53], GoogleNet [55], ResNet [33], RCNN [31], YOLO, SSD [42].

[1] https://en.wikipedia.org/wiki/Transfer_learning

2. Trackers: MOT [59], MDP [64], SORT [61], DORT

3. Data: ImageNet [27], MNIST [24], COCO [40], OpenImages [36], CIFAR [4]

4. Robotic Datasets: KITTI [29], VOT2016 [27], UAV123 [17], OTB100 [62], UAVDT [16], Caltech [3], CARPK [58], ALOV300++ [1], PETS [45]

5. Languages: Python, C/C++, Matlab

6. Frameworks[2]: Tensorflow, Keras, Pytorch, DarkNet, OpenNN, MatConNet

4.1 Deep Learning Detection

This thesis has interest in setting up a tracking system using Deep Networks where in recent times (after 2017) plenty of papers been published. Here I summarize briefly latest network architectures in demand describing the performances according to Object Detection Metrics especially mean-Average-Precision(mAP) & frames-per-second(fps). Refer this paper to have an idea[3].

RetinaNet [39] Facebook's AI research is a single, unified network composed of a backbone network and two task-specific subnetworks. The backbone Feature Pyramid Network(FPN) is responsible for computing a convolutional feature map over an entire input image and is an off-the-self convolutional network & achieves mAP: 33 to 40, fps: 70ms to 200ms. *DetNet* [37], *NAS-FPN* [30] both advances the RetinaNet backbone; ts stage 1,2,3,4 the same as original ResNet-50, spatial resolution FPN used & observed to be better than ResNet-101 and Mask RCNN, whereas other one has search tendency to learn a RNN controller and it is trained using the Proximal Policy Optimization. This network is applied for high detection accuracy and fast inference. *DeNet* [57] uses encoder-decoder estimation network that is helpful in estimating the density map for the areas where individuals cannot be segmented due to high crowdedness. Thus this net has proven to have Lower Mean Absolute Error but high-quality density maps without losing resolution. *CenterNet* [28] modifies both ResNets and DLA-34 using deformable

[2] https://en.wikipedia.org/wiki/Comparison_of_deep-learning_software

[3] https://www.researchgate.net/publication/237749648_Performance_Evaluation_of_Object_Detection_and_Tracking_Systems

convolution layers and use the Hourglass network as it is. This net is end-to-end differentiable, more accurate than other bounding box detectors as it uses key-point estimation to find center points and regresses to all other object properties, such as size, 3D location, orientation, and even pose achieving 45.1 mAP with multi-scale testing at 1.4 FPS. *PoolNet* [41] has U-shape architecture where it first builds a global guidance module, feature aggregation module, then Joint Training is done. Due to Joint Training this network boosts MAE[4] metrics. It also got the advantages in high-level semantic features to be progressively refining & it is the first paper aiming at pooling-based modules (achieves 30 FPS).

PV-RCNN [52] is observed to be employed for Point Cloud Object Detection, Feature Encoding and Proposal Generation, Scene Encoding viaVoxel Set Abstraction, keypoint to grid RoI Feature Abstraction & has applications in conventional pooling as the RoI-grid feature points encode much richer context. It gives high-quality proposals of 3D voxel CNN and the flexible receptive fields of the PointNet-based networks. *LapNet* [22] is an anchor based detector, trained end-to-end without any sampling strategy & PONO Anchor assignments. Additional weights at inference makes good compromise in performance and computational speed, simple and generic with speed 50ms to 75ms. *ResNeST* [66] follows resnet with Radix-major Split-Attention proves to have more mAP than Faster-RCNNs and Cascade-RCNNs. *FCOS* [56] if based on ResNeXt-64x4d-101, deformable conv - 49.0 in AP on COCO test-dev & has benefit of eliminating pre-defined set of anchor boxes, complicated X computations avoiding all hyper-parameters.

RefineDetLite [23], *Instance-Aware-Context-Focused-Memory Efficient Weakly Supervised Network* [50], *Gaussian Mixture Probability MOT + Occlusion Group Scheme* [54], *SimCLR* [25] are known for avoiding FLOPs with three orthogonal training strategies—IOU-guidedloss, learnable Concrete DropBlock while devising a memory efficient sequential batch back-propagation for multiple instance self training, performing track merging-occlusion group energy minimization of occlusion ratio between visual objects with sum-of-intersection-over-area (SIOA) instead of IoU and Contrastive Learning by Data augmentations, from larger batch sizes with more training respectively.

Initially, sliding window algorithm was used for object detection. Later on

[4] https://en.wikipedia.org/wiki/Mean_absolute_error

turning Fully Connected Layers into CN Layers gave scope to apply sliding windows over CNNs, but notably bounding box fixing is not accurate. After these, CNN+Search sequence based algorithms which doesnot output the proper bounding box of an object nor the shaped object with the ground truth boxes expectation. Then came YOLO(2015) in which ground truth bounding box is a hand-labelled one from testing set that specify where the object is in the image. Bounding boxes[5], Intersection-over-Union, Non-max Suppression, Anchor boxes & Region Proposals are the regular used terms while explaining YOLO versions.

4.2 Choice of YOLO or RetinaNet

YOLO has proven to be better, faster and more accurate tracking scheme and stands tall as milestone in object detection research that has four versions till date. YOLOv4 is the most recent one. *Joseph Redmon*[6] is the father of YOLO.

YOLOv1 [47] suggests to have unified network to perform all at once. Also, end-to-end training network can be achieved. The model has 24 conv layers followed by 2 fully connected layers. YOLOv2 [48] used a custom deep architecture 19-layer network supplemented with 11 more layers for object detection. YOLOv2 often struggled with small object detections. This was attributed to loss of fine-grained features as the layers downsampled the input. It used an identity mapping, concatenating feature maps from from a previous layer to capture low level features. However, it lacks residual blocks, no skip connections and no upsampling. YOLOv3 uses all of these.

YOLOv3 [49] uses Darknet, which originally has 53 layer network trained on Imagenet. For the task of detection, 53 more layers are stacked, so in total 106 layer fully convolutional underlying architecture for YOLOv3, refer figure 4.1. This is the reason behind the slowness of YOLOv3 compared to YOLOv2. Alexey Bochkovskiy and team, built the YOLOv4 [20]as Windows version. Its advantages: It enables anyone with a 1080 Ti or 2080Ti GPU to train a super fast & accurate object detector. The influence of "Bag-of-Freebies" & "Bag-of-Specials" object detection methods during detector training has been verified.

[5] http://christopher5106.github.io/object/detectors/2017/08/10/bounding-box-object-detectors-understanding-yolo.html

[6] Website Darknet:https://pjreddie.com/

Modified methods, including CBN (Cross-iteration batch normalization), PAN (Path aggregation network), etc., are now more efficient and suitable for single GPU training. It obtained an AP value of 43.5 (65.7% AP50) on the MS COCO dataset, and achieved a real-time speed of 65 FPS on the Tesla V100, beating the fastest and most accurate detectors in terms of both speed & accuracy. Compared with YOLOv3, AP and FPS have increased by 10% and 12%, respectively.

So, let's understand how YOLO works:

1. Take an image (say 100X100). Divide image to grids (say 3 by 3). Take the image classification & localization algorithm & run it on each of 9 grids.

2. So for each grid, specify a label Y (a 8D vector), y= $\begin{bmatrix} p_c & b_x & b_y & b_h & b_w & c_1 & c_2 & c_3 \end{bmatrix}^\top$. Where p_c is 0 or 1 depends on object is present or not. b_x, b_y, b_h, b_w describes the bounding box's midpoint, height, width & c_1, c_2, c_3 gives type of class that object belongs (if we consider #3 classes). Thus you get training labels.

3. Now if input image is 100X100X3, then choose conv layers, max pool layers and so on, so that you map this image to 3X3X8 output volume. Use Back propogation[7] to train NN to map from input to this output volume.

4. You will get the predictions. So at test time, what you do is you feed an input image X and run forward propogation until you get this output Y

Note: The advantage is this NN outputs precise bounding boxes. So each object, even if the objects spends multiple grid cells, that object is assigned only to one of the 9 grids. For much finer 19X19 grid, the chance of an object of two midpoints of objects appearing in the same grid cell is smaller. Also, note we won't run the same algorithm 3X3 times, this is one single conv implementation, where you use one conv net with a lot of shared computation between all the computations needed for all of your 3X3 grid cells.

YOLO algorithm, which is constant popularity is because this is a conv implementation, it actually runs very fast. So this works even for real time object detection. Also, *for training*, do consider the idea of specifying the bounding box b_x, b_y -lies between 0 and 1, where asb_h, b_w could be > 1. Choose a IoU

[7]Back-propagation: https://en.wikipedia.org/wiki/Backpropagation

function over 0.5, a proper *non-max suppression coefficient* (suppressing multiple detections for each object, usually discard all boxes $p_c <= 0.6$), implementing non-max suppression on multiple object classes c_1, c_2, c_3 & if a grid cell wants to detect multiple class objects (say #2) use *anchor boxes* (where shape of the 'Y' vector now becomes 3X3X8X2 for considering 2 anchor boxes - $\left[p_c\ b_x\ b_y\ b_h\ b_w\ c_1\ c_2\ c_3\ p_c\ b_x\ b_y\ b_h\ b_w\ c_1\ c_2\ c_3\right]^\top$, first 8 elements belong to anchor box1 & next 8 belongs to anchor box2). Passing through the conv layers make predictions, then *for testing* you will get 2 anchor boxes for each grid cell, getting rid of low probability predictions & running a non-max suppression gives the accurate detection outputs. Refer Coursera online for more explanations [7].

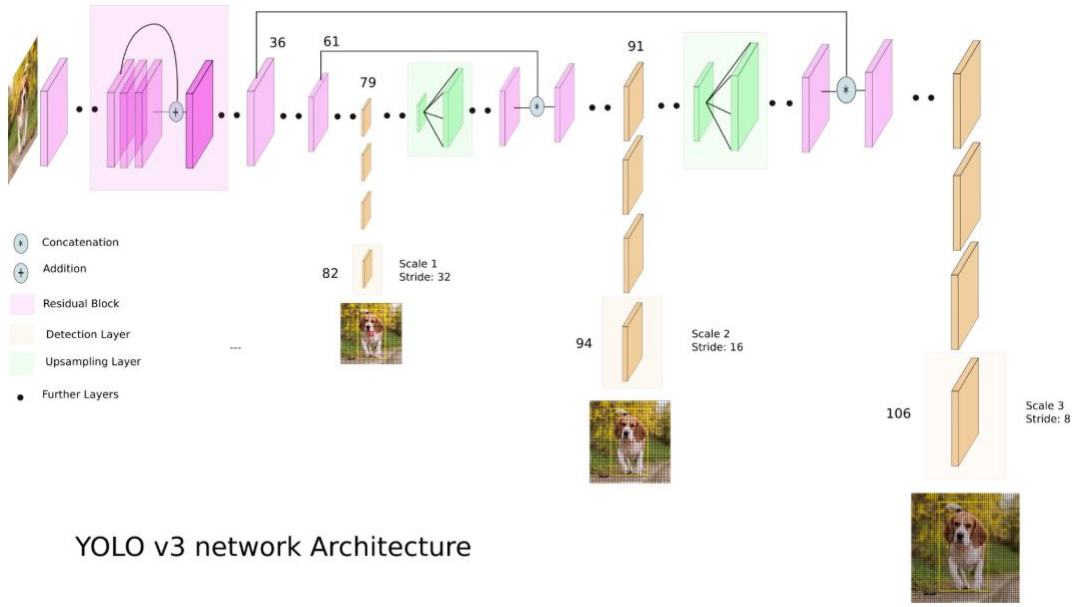

Figure 4.1: YOLOv3 Architecture for Detections at 3 Different scales

YOLOv3 boasts of residual skip connections & upsampling, refer figure 4.1. The most important feature of v3 is that it makes detections at three different scales. YOLO is a fully conv net & the detection is done by applying 1 x 1 detection kernels on feature maps of three different sizes at three different places in the network. The shape of the detection kernel is 1 x 1 x (B x (5 + C)), refer figure4.2. Here B = number of bounding boxes a cell on the feature map can predict, "5" is for the 4 bounding box attributes and one object confidence, and C = number of classes. In YOLO v3 trained on COCO, B = 3 and C = 80, so the

kernel size is 1 x 1 x 255. The feature map produced by this kernel has identical height and width of the previous feature map, and has detection attributes along the depth as described above. Stride of the network or a layer is defined as the

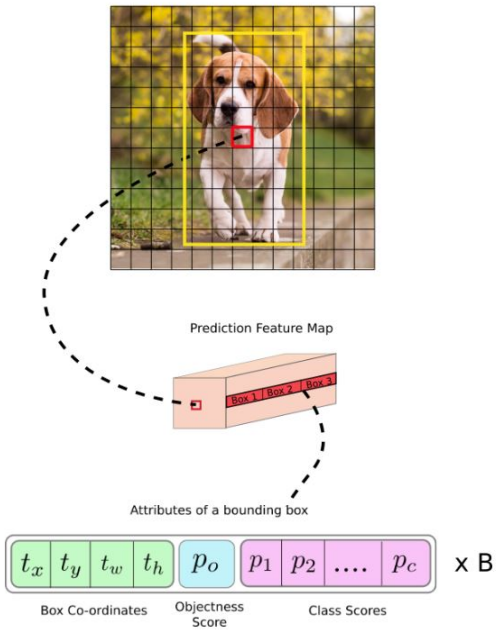

Figure 4.2: Image grid (red) responsible for Object Detection

ratio by which it downsamples the input. Assume an input image of size 416 x 416. YOLOv3 makes prediction at three scales, which are precisely given by downsampling input image by 32, 16 and 8 respectively. Refer figure4.1. Note: content, explanations related to figure 4.1 and 4.2 are taken from [19]. First detection is made by the 82nd layer. For the first 81 layers, the image is down sampled by the network, such that the 81st layer has a stride of 32. For an input image of 416 x 416, the resultant feature map is size 13 x 13. One detection is made here using the 1 x 1 detection kernel, giving us a detection fmap of 13 x 13 x 255. Then, the feature map(fmap) from layer 79 is subjected to a few conv layers before being up sampled by 2x to dimensions of 26 x 26. This feature map is then depth concatenated with the feature map from layer 61. Like before, a few 1 x 1 convolutional layers to fuse the features from the earlier layer (61). Then, the second detection is made by the 94th layer, yielding a detection feature map of 26 x 26 x 255. Repeat again, the fmap from layer 91 is subjected to few conv layers before being depth concatenated with a fmap from layer 36. Like before, a

few 1 x 1 conv layers follow to fuse the information from the previous layer(36). We make the final one at 106th layer, yielding feature map 52 x 52 x 255 size.

The upsampled layers concatenated with the previous layers help preserve the fine grained features which help in detecting small objects. It is better strategy at detecting smaller, medium & larger objects. Choice of anchor boxes, YOLOv3 in total uses 9 anchor boxes, three for each scale. More bounding boxes per image: For an input image of same size, YOLOv3 predicts more bounding boxes(10x) than YOLOv2. You could easily imagine why it's slower than YOLOv2. Softmax is removed, as they only work for detections of mutually exclusive classes. YOLOv4 [20] also got Weighted-Residual-Connections (WRC), Cross-Stage-Partial-Connections (CSP), Cross mini-Batch Normalization (CmBN), Self-adversarial-training (SAT), Mish-activation, Mosaic data augmentation, DropBlock regularization, CIoU loss features improving YOLOv3.

Now what about the RetinaNet?

Detectron [32] is Facebook AI Research's software system that implements state-of-the-art object detection algorithms, including Mask R-CNN. It is written in Python and powered by the Caffe2[8] DL framework. All training annotations must be in the COCO format. Detectron currently only runs on the GPU, one can train your model, and then modify all of the inference code in order to load your dataset into the evaluation scripts. Detectron includes implementations of the following object detection algorithms: Mask R-CNN, RetinaNet, Faster R-CNN, RPN, Fast R-CNN, R-FCN using the following backbone network architectures: ResNeSt(50,101,152), ResNet(50,101,152), Feature Pyramid Networks & VGG16. Detectron2 [63] is powered by the PyTorch deep learning framework (new vision library) that allows us to easily use and create object detection, instance segmentation, keypoint detections. It has a simple, modular design that makes it easy to rewrite a script for another data-set. On top of Detectron, it includes more features such as panoptic segmentation, densepose, Cascade R-CNN, rotated bounding boxes, etc. It trains much faster.

[8] https://caffe2.ai/docs/getting-started.html?platform=windows&configuration=compile

Now, let's understand how RetinaNet architecture works:

RetinaNet [39] is one of the best one-stage object detection models (included in Detectron2) that work well with dense and small scale objects. RetinaNet has been formed by making two improvements over existing single stage object detection models - Feature Pyramid Networks (FPN) [38] and Focal Loss [39]. Before knowing RetinaNet's architecture, let's first understand FPN. Featurized image pyramids have been used to detect objects with varying scales in an image. This means one would take an image and subsample it into lower resolution and smaller size images (thus, forming a pyramid). Hand-engineered features are then extracted from each layer in the pyramid to detect the objects. This makes the pyramid scale-invariant. But, this process is compute and memory intensive. With the advent of deep learning, these hand-engineered features were replaced by CNNs. Later, the pyramid itself was derived from the inherent pyramidal hierarchical structure of the CNNs. In a CNN architecture, the output size of feature maps decreases after each successive block of convolutional operations, and forms a pyramidal structure. FPN creates an architecture with rich semantics at all levels as it combines low-resolution semantically strong features with high-resolution semantically weak features. This is achieved by creating a top-down pathway with lateral connections to bottom-up convolutional layers. RetinaNet incorporates FPN and adds classification and regression subnetworks to create an object detection model.

There are four major components of a RetinaNet model(refer figure 4.4):

1. *Bottom-up Pathway* - The backbone network (e.g. ResNet) which calculates the feature maps at different scales, irrespective of the input image size or the backbone.

2. *Top-down pathway and Lateral connections* - The top down pathway up-samples the spatially coarser feature maps from higher pyramid levels, and the lateral connections merge the top-down layers and the bottom-up layers with the same spatial size.

3. *Classification subnetwork* - It predicts the probability of an object being present at each spatial location for each anchor box and object class.

4. *Regression subnetwork* - It's regresses the offset for the bounding boxes from the anchor boxes for each ground-truth object.

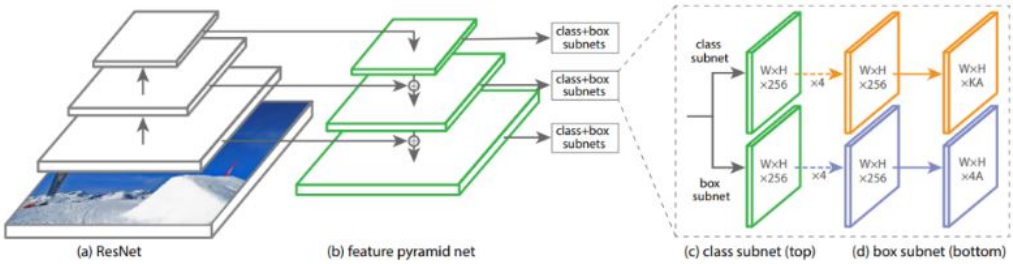

Figure 4.3: RetinaNet Architecture with Feature Pyramid Backbone

Focal Loss (FL) is an modified Cross-Entropy Loss (CE) and is introduced to handle the class imbalance problem with single-stage object detection models. Single Stage models suffer from a extreme foreground-background class imbalance problem due to dense sampling of anchor boxes (possible object locations) [39]. In RetinaNet, at each pyramid layer there can be thousands of anchor boxes. Only a few will be assigned to a ground-truth object while the vast majority will be background class. These easy examples (detections with high probabilities) although resulting in small loss values can collectively overwhelm the model. Focal Loss reduces the loss contribution from easy examples and increases the importance of correcting missclassified examples. Explanation found here [12].

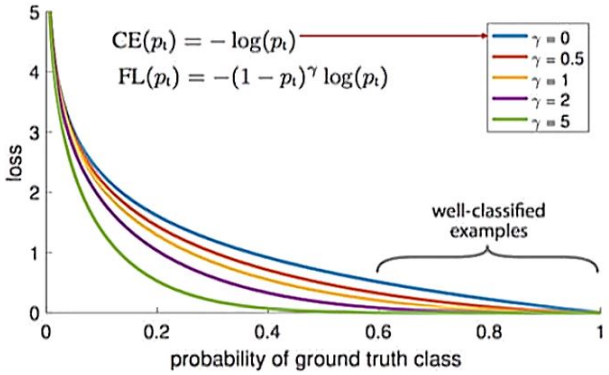

Figure 4.4: RetinaNet - Focal Loss function for the focus on hard examples

Focal loss function is formulated as: FL(p_t)= -(1-p_t)$^{\gamma}$ log(p_i) where 'p' is the probability of the ground truth classes. This focal loss is used to train the network

for better box detection accuracy, keeping speed intact. See the self explanatory graph figure4.4. The procedural explanation is taken from this link[9].

4.3 Depth Estimation

Great! we are able to use the tracking system developed for tracking the moving target robot in the 3D environments including time of occlusion setup. But it is also important aspect to know the depth of the target and the occlusion from the aerial robot, refer figure.3.2. Also several investigations are done as part of this thesis work. Aside of the classical mathematical calculations for the depth measurements, there exists a start-of-the-art Joint Monocular 3D Vehicle Detection.

4.3.1 DL Approach (ex. Joint Monocular 3D - Detection)

Joint Monocular 3D Detection [34] emphasizes on vehicle 3D extents and trajectories are critical cues for predicting the future location of vehicles and planning future agent ego-motion based on those predictions. In this paper, authors propose a novel online framework for 3D vehicle detection and tracking from monocular videos. The framework can not only associate detections of vehicles in motion over time, but also estimate their complete 3D bounding box information from a sequence of 2D images captured on a moving platform. And this method leverages 3D box depth-ordering matching for robust instance association and utilizes 3D trajectory prediction for re-identification of occluded vehicles. It is stated as the design of a motion learning module based on an LSTM[10] for more accurate long-term motion extrapolation. Experiments on simulation, KITTI, and Argoverse datasets show that our 3D tracking pipeline offers robust data association and tracking. On Argoverse, author's image based method is significantly better for tracking 3D vehicles within 30m than the LiDAR-centric baseline methods.

[9] RetinaNet https://developers.arcgis.com/python/guide/how-retinanet-works/
[10] https://en.wikipedia.org/wiki/Long_short-term_memory

4.3.2 Standard Approach

Following the available source on Real time Depth Estimation and Obstacle Detection from Monocular Video [60], we first describe the underlying perspective projection and the model used for depth computation. For a point $\mathbf{X}=(X,Y,Z)^T$ in 3D space we obtain the image point $\mathbf{x}=(x,y)^T$ by perspective projection:

$$x = \frac{f}{Z}\begin{bmatrix} X \\ -Y \end{bmatrix} \tag{4.1}$$

The camera translation in depth between consecutive frames is known from inertial sensors. In practice the relative depths on obstacle surfaces are small compared to the distance between obstacle and camera such that this assumption is a good approximation. Then, camera translation in depth between time t and t+τ is T(t,τ) leading to $X(t+\tau) = X(t) + T(t,\tau)$. The camera translation and rotational velocity is $\dot{T}(t)$ and $\dot{\Omega}(t)$ respectively. The image velocity field mentioned. Therefore resolving above using an explicit or implicit integration method and under the given conditions the scene depth can be estimated solely by estimating the scaling factor of image regions. The transformation of an image point for a pure translation using first eqn. becomes, $x(t+\tau)$ where,

$$x(t+\tau) = \frac{Z(t)}{Z(t)+Tz(t,\tau)}\ \frac{f}{Z(t)}\ \cdot \begin{bmatrix} X(t) \\ Y(t) \end{bmatrix} + \frac{f}{Z(t)+Tz(t,\tau)}\ \cdot \begin{bmatrix} TX(t,\tau) \\ TY(t,\tau) \end{bmatrix}$$

Use absolute image coordinates and not velocities for computation. With a correctly given vehicle translation and displacement of image points, scene depth can be directly calculated over large time scales. Therefore multiple observations in an image region can be used to solve an over–determined equation system for the scaling factor and the translation. The key for depth reconstruction is to use the scale s(t, τ) directly obtained by the used region tracking over multiple frames to calculate scene depth(refer eqn.5 of the paper):

$$d \equiv Z(t) = \frac{s(t,\tau)}{1-s(t,\tau)}\ Tz(t,\tau)$$

Distance histogram, Error in depth estimation and Obstacle Detection by Hypothesis Verification are also explained in the paper cited [34].

4.4 Conclusion

Finally, considering the time occlusion formula t_{occ} to feed to the control which helps aerial robot's control and state of the robot parameters to change & still keep tracks (with state-of-the-art tracking algorithms YOLO and RetinaNet) of occlusion and target as per described Methodology2. As per the object detection metrics, mAP(Mean Average Precision) for detection accuracy and fps (frames per second) for the speed to keep track the moving targets are standard parameters to be studied during implementation.

Chapter 5

Implementation

'Try and try until you succeed, drop and drop fills the ocean'

-anonymous

With reference to Time of Occlusion concept developed before (in Chapter3, I realised the State-of-the-art YOLO(YOLOv3) and RetinaNet(Detectron2), which are expected to be future milestones for Aerial Robotic Research. Tensorflow[1] is a free machine learning open-source library for dataflow and differentiable programming across a range of tasks and neural networks. Keras[2] is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow. Here Keras is used for Custom Object Detection. PyTorch[3] is also an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab. Caffe2[4] is a open source deep learning framework which is written in C++, with a Python interface. This supports many different types of deep learning architectures geared towards image classification and image segmentation. It supports CNN, RCNN, LSTM and fully connected neural network designs which where used to develop Detectron2. In March 2018, Caffe2 was merged into PyTorch.

[1] https://www.tensorflow.org/install
[2] https://keras.io/getting_started/
[3] https://pytorch.org/get-started/locally/
[4] https://caffe2.ai/docs/tutorials

5.1 YOLO Implementation + Results

YOLOv3 implementation is done over this existing structure:

1. *Dependencies*: TensorFlow 1.15, Keras 2.2.4, Numpy 1.16, Opencv 3.6.9, python 3.2.4.17

2. *Model hyperparameters*: batch size: 8 (can be 8 or 32). Batch Norm Epsilon = 1e-5, leaky ReLU=0.1, anchor boxes - 9 no.s, epoches: CPU used

3. *Model definiton with Batch norm*: Fixed padding, Darknet-53 residual connections, 72 Conv.Layers, Upsample layer, Non-max suppression

4. *Utility functions* help us load images as NumPy arrays, load class names from the official file and draw the predicted boxes

5. *Converting weights to Tensorflow format*: to load the official weights, tf.assign operations returns a list of assign operations.

For each cell in the feature map the detection layer predicts $n_{anchors} * (5 + n_{classes})$ values using 1x1 convolution. For each scale we have $n_{anchors} = 3.5 + n_{classes}$ means that respectively to each of 3 anchors we are going to predict 4 coordinates of the box, its confidence score and class probabilities. In order to concatenate with shortcut outputs from Darknet-53 before applying detection on a different scale, then upsampling the feature map using nearest neighbor interpolation. Also, to avoid multiple boxes for one object we use non-max suppression for each class. I referred this link [11]

Here in this implementation I used *"Transfer Learning"* Technique[5] (pre-trained network, annotations and weights) in the above task. But actual procedure to create Custom Object Detection is given in SIX STEPS below:

1. *Creating custom/existing Dataset*: Here I used COCO Imageset(80 classes).

2. *Drawing Annotations*: Convert image data into text annotations files.

3. *Downloading Weights* Downloading weights from YOLO darknet and network config. file (These gives the designed network layout).

[5]that shortcuts much of this by taking a piece of a model that has already been trained on a related task and reusing it in a new model

4. *Training phase*: Train the network (part by part by using GPU or completely using CPU) - by selecting the epoch, batch sizes and cost(gives model performance) of YOLO network.

5. *Export Inference Graph*: From new model, thus we get YOLO trained config files. (Optional: Convert them to Keras model for custom detection). Run to we convert image data to tensors.

6. *Testing Phase*: Detect custom Object Detection in real time, Either use image / computer screen / Webcam as input test images for Detection by Frames

Figure 5.1: Output of YOLOv3 Objection Detection & Tracking using Webcam

YOLOv3 Object Detection-1 (webcam)[6]: @2 fps
YOLOv3 Object Detection-2 (webcam)[7] : @6 fps
YOLOv4 Object Detection-3 (webcam)[8] (CPU) : @0.2 fps

[6]The video recorded output in my PC (CPU) https://youtu.be/J-xz2tQTK1c
[7]The video recorded output in LAAS PC (CPU): https://youtu.be/ytDOpJ9FOmc
[8]The video recorded output in LAAS PC: https://youtu.be/RpzwnDiODpA

YOLOv4 Object Detection-4 (webcam)[9] (GPU) : @4.8 fps
Please refer table for specifications 6.1

5.2 RetinaNet Implementation + Results

Detectron2 [8] is a ground-up rewrite of Detectron that started with maskrcnn-benchmark. The platform is now implemented in PyTorch. With a new, more modular design, Detectron2 is flexible and extensible, and able to provide fast training on single or multiple GPU servers. Detectron2 includes high-quality implementations of state-of-the-art object detection algorithms, including DensePose, panoptic feature pyramid networks and numerous variants of the pioneering Mask R-CNN model family. Its extensible design makes it easy to implement cutting-edge research projects without having to fork the entire codebase. So, for RetinaNet implementation, I referred [63], its structure:

1. *Dependencies:* Linux, $Python = 3.6, PyTorch = 1.4$, torchvision, OpenCV
2. *gcc and CUDA*: gcc & g++ over 5 are needed, CUDA = 10.1, CuDnn = 7.0
3. *Other tools*: pycoco tools, PythonAPI, cython, Virtual Environment

RetinaNet is part of Detectron2 that is referred to as a software system with group of projects. It is intended to install & implement the entire detectron2 structure. Having said that, RetinaNet is now referred to as Detectron2 and vice-versa. Procedure to follow **Build and install** Detectron2:

1. create a virtual environment called *myenv*. Clone the repository detectron2
2. install the detectron2 package with specified root structure
3. install pytorch, opencv over 4, pycoco tools, cython, *gcc* 4.5 & pillow
4. run the inference demo with the pre-trained models using webcam

Here in the last step, user is allowed to choose the network architecture along with the backbone to be included while training to obtain results. Finally, after installing the Detectron2, as a result, for now as I could find proper CUDA version

[9]The video recorded output in LAAS PC: https://youtu.be/06AHNylwcoo

to install for my GPU Nvidia GeForce 800M in my PC and my LAAS Work PC GPU Quadro 1200M, this algorithm/network installation gave the below window.

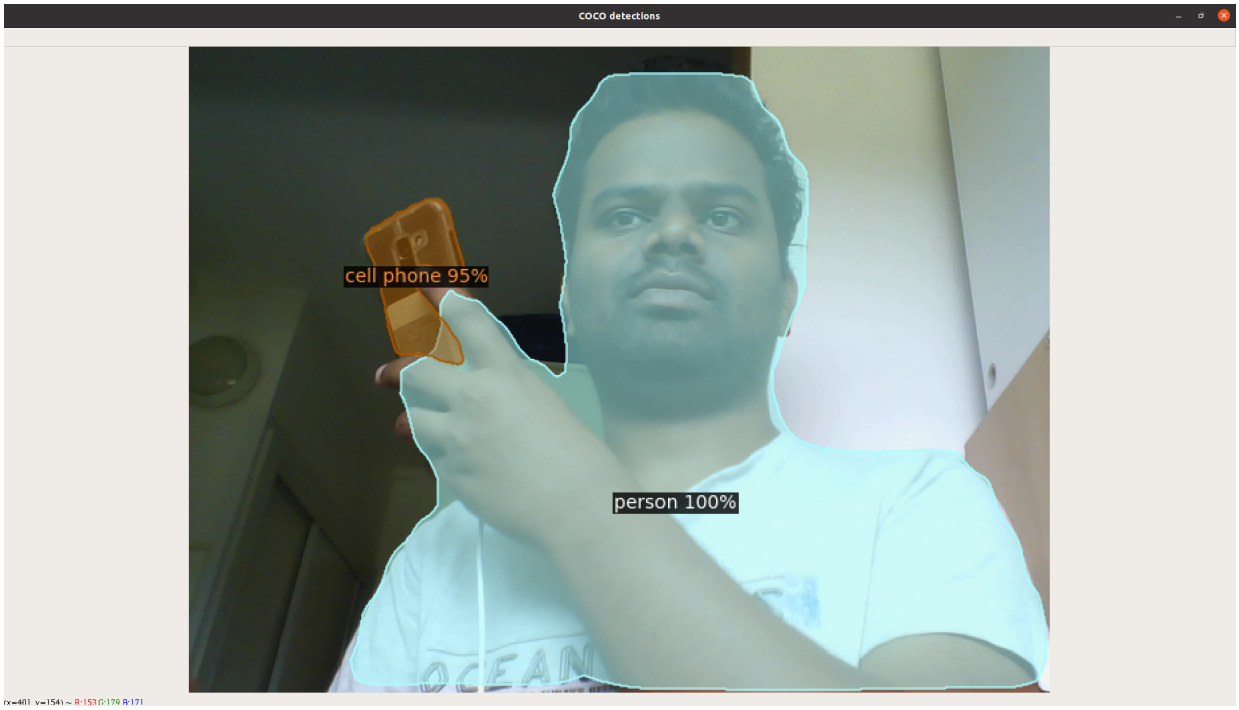

Figure 5.2: Output of Detectron2 - RetinaNet using Webcam

Detectron2 Object Detection-1 (webcam)[10]: @2.35s/iter
Detectron2 Object Detection-2 (webcam)[11]: @1.93s/iter
Please refer table for specifications 6.1

5.3 Depth Estimation

In short, to detect the distance of target from camera, triangle similarity [13] technique can be used. Let a target is at a distance D (in centimetres) from camera and the target's actual height is H (I have assumed that the average height of target in centimetres). Refer figure 5.3. Using the object detection techniques, we can identify the pixel height P of the person using the bounding box coordinates.

[10]The video recorded output in LAAS PC (CPU) https://youtu.be/rZjaWulg4lQ
[11]The video recorded output in LAAS PC (GPU) https://youtu.be/LFksvpn_jSs

Using these values, the focal length of the camera can be calculated using formula: Eq 1: F = (P x D) / H. After calculating the focal length of the camera, we can use the actual height H of the target, pixel height P of the person and focal length of camera F to calculate the distance of the target from camera. Distance from camera can be calculated using: Eq 2: D' = (H x F) / P. Repeat this for finding depth of occlusion, see [13] for more information. Now that we know the depth of the target & occlusion from camera, we can move on to calculate the distance between two them in a environment. There can be 'n' occlusions detected in a environment. So the Euclidean distance is calculated between the mid-point of the bounding boxes of all the occlusions & targets detected. By doing this, we get our x & y values. These pixel values are expressed in cms using Eq 2.

We have the x, y and z coordinates for every target or a occlusion in cms. Euclidean distance between them detected is calculated using the (x, y, z) coordinates. If the distance is (say) less than 1 metre or 100cms, a red bounding box is displayed around them indicating that they are closer to each other. This Depth estimation is actually a concept developed social distancing during COVID crisis, I referred this link [13] for depth estimation. Thus, the same logic can be adopted to our Depth Estimation in the future.

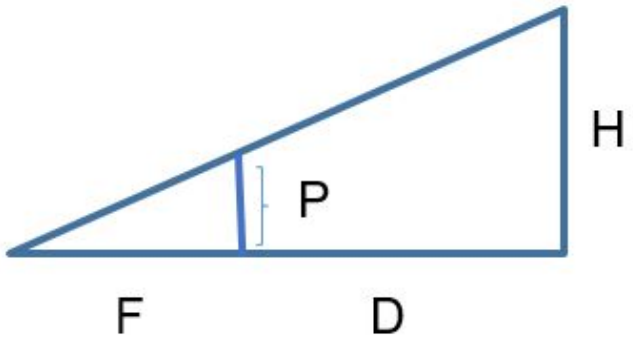

Figure 5.3: Triangular Similarity for Depth Processing

Chapter 6

Results and Discussion

6.1 Results

'Think Deep, or taste not the Pierian spring'

-Alexander Pope

The results of YOLO and RetinaNet are described in subsections 5.1 and 5.2 respectively. Here is the tabulated results and specifications with PC configurations used as part of this thesis work. Note that due to COVID lockdown and work from home, the camera used here is a webcam, but can be replaced with the connection of the camera mounted through a USB port on the aerial robot.

Table 6.1: Object Tracking Results and Analysis

Algorithm	CPU or GPU	Efficiency
YOLOv3	Intel® Core™ i5-4210U CPU(Ubuntu18.04)	40.1AP / @2 fps
YOLOv3	Intel® Core™ i7-7820U CPU(Ubuntu16.04)	41.4AP / @6 fps
YOLOv4	Intel® Core™ i7-7820HQ CPU(Ubuntu20.04)	40.1AP / @0.2 fps
YOLOv4	Intel® Core™ i7-7820HQ GPU(Ubuntu20.04)	42AP / @4.8 fps
Detectron2	Intel® Core™ i7-7820HQ CPU(Ubuntu20.04)	2.35s/iter
Detectron2	Intel® Core™ i7-7820HQ GPU(Ubuntu20.04)	1.93s/iter

6.2 Discussions

Objection Detection metrics used in this thesis work are: mAP and fps to test the performance of the algorithms. Notably, for the efficiency test, YOLOv3 acquires 40mAP and 2fps on i5 machine on Ubuntu18.04 whereas the same YOLOv3 tested on i7 machine on Ubuntu16.04 gives 41mAP and 6fps.

RetinaNet gives GPU (CUDA Runtime Error) as for now no proper CUDA, CuDNN version tools & drivers are not found. *RetinaNet uses a CUDA version over 9.0, where my PC is not compatible with only CUDA under 8.0.* My LAAS PC can be compatible with CUDA version 9.0 and currently under test. Nvidia CUDA Compatability comparison is given here [14].

Also note that Detectron2(RetinaNet) as described already its a Facebook AI Detection Software system, Facebook team is currently recycling the repositories of the RetinaNet. So, finally the task of the thesis is achieved considering YOLOv3 as the tracking system, but for the efficiency test / results comparison RetinaNet has to be functionable and use the GPU (as CUDA is a mandatory to install Detectron2). After this learning how to use GPU, both YOLO and RetinaNet are trained on GPU. Bounding boxes are displayed around target and occlusion detected, depth is fed as another input for controller for robot 3D motion.

Here, Joint Monocular 3D Tracking System is explained as one of the state-of-the-art established algorithms which uses depth information while tracking & has no concern to implement it as part of the thesis work. YOLOv4 is the referred to as the improvements over YOLOv3 but has no mandatory concern to implement it as part of the thesis work as it is a latest released paper (23 Apr 2020) & expected to have some fixtures in the YOLOv4 software.

Chapter 7

Conclusion

'Artificial Intelligence is a tool, not a threat'

-Rodney Brooks

7.1 Conclusion

To summarize, considering a quadrotor with downward looking camera in a 3D environment, we developed a methodology to avoid occlusion while tracking. It is to express the concept of time to occlusion which is the time before loosing visibility of the target. Then, we proposed a simple yet sufficient mathematical model expressing this quantity, which is fed as input to the controller as an additional information. In order to implement this, it is obvious to use CNN algorithms because as observed tuning the network parameters is easy. So, in process of choosing the established CNN networks, reviewed the state of the art in this field as described in section4.1 and selected few of them which are YOLO, RetinaNet, 3D Joint Monocular Depth Tracking.

As far as the implementation & results are concerned the novel tasks of find the time of occlusion and a tracking system for the aerial robotic setup for the described configurations are fully achieved considering YOLO as tracking system. RetinaNet can be used and compared to YOLO which makes a choice for deployment of a tracking system on aerial robot to move in the 3D world environments,

however both has the profound impact in their own ways as explained. The higher the values of mAP and fps the better the performance of tracking system which are used for the efficiency test of our algorithms.

It goes without saying that the COVID-19 situation jeopardized the work because it was harder for the supervisors to follow & advise me.

7.2 Future Work

My thesis ends on 31st of July. So, I will continue to learn how to include the depth information and transforming the existing tracking system as a depth based tracking system. We plan to collect the small dataset, apply CNN using Intel NUC and then compare their performance. This is left over future work. Moving on, this mathematical model & tracking system will be implemented on a robotic platform, using ROS or GenoM, and Gazebo simulations before going for real experiments.

Bibliography

[1] Alov300++ dataset. http://alov300pp.joomlafree.it/.

[2] B-splines — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/B-spline.

[3] Caltech pedestrian detection benchmark. http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/.

[4] Cifar-10. https://www.cs.toronto.edu/~kriz/cifar.html.

[5] Control architecture2 — GitHub. https://github.com/vamshikodipaka/Probabilistic-Robotics/blob/master/qu6.JPG.

[6] Coursera — Coursera online, robotics: Aerial robotics. https://www.coursera.org/learn/robotics-flight/home/welcome.

[7] Coursera — Coursera online, yolo algorithm: Deeplearning.ai. https://www.coursera.org/learn/convolutional-neural-networks/lecture/fF3O0/yolo-algorithm.

[8] Detectron2 — Facebook, : A pytorch-based modular object detection library. https://ai.facebook.com/blog/.

[9] Model predictive controller — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Model_predictive_control.

[10] Motion detector — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Motion_detector.

[11] Pylessons — YOLOv3, demonstrating yolo v3 object detection with webcam. https://pylessons.com/YOLOv3-WebCam/.

[12] Retinanet — Facebook, : How retinanet works. https://developers.arcgis.com/python/guide/how-retinanet-works/.

[13] Social distancing — Github, social-distance-detection-using-opencv. https://github.com/Subikshaa/Social-Distance-Detection-using-OpenCV.

[14] System-compatability-comarision — Nvidia, nvidia-gpu. https://developer.nvidia.com/cuda-gpus.

[15] Trajectory replannning strategies — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Motion_planning.

[16] Uav benchmark: Object detection and tracking. https://www.researchgate.net/publication/324167192_The_Unmanned_Aerial_Vehicle_Benchmark_Object_Detection_and_Tracking.

[17] Uav dataset. https://uav123.org/.

[18] Video tracking — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Video_tracking.

[19] Yolov3 — Towards Datascience, what's new in yolo v3? https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b.

[20] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[21] Claudette Cédras and Mubarak Shah's. "motion-based recognition: a survey". *Image and Vision Computing; Elsevier Science B.V*, March 1995.

[22] Florian Chabot, Quoc-Cuong Pham, and Mohamed Chaouch. Lapnet : Automatic balanced loss and optimal assignment for real-time dense object detection, 2019.

[23] Chen Chen, Mengyuan Liu, Xiandong Meng, Wanpeng Xiao, and Qi Ju. Refinedetlite: A lightweight one-stage object detection framework for cpu-only devices, 2019.

[24] Feiyang Chen, Nan Chen, Hanyang Mao, and Hanlin Hu. Assessing four neural networks on handwritten digit recognition dataset (mnist), 2018.

[25] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

[26] X. Cheng, M. Honda, N. Ikoma, and T. Ikenaga. Anti-occlusion observation model and automatic recovery for multi-view ball tracking in sports analysis. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1501–1505, 2016.

[27] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei Fei Li. Imagenet: a large-scale hierarchical image database. pages 248–255, 06 2009.

[28] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection, 2019.

[29] Andreas Geiger, P Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: the kitti dataset. *The International Journal of Robotics Research*, 32:1231–1237, 09 2013.

[30] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection, 2019.

[31] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013.

[32] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. https://github.com/facebookresearch/detectron, 2018.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[34] Hou-Ning Hu, Qi-Zhi Cai, Dequan Wang, Ji Lin, Min Sun, Philipp Krähenbühl, Trevor Darrell, and Fisher Yu. Joint monocular 3d vehicle detection and tracking, 2018.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[36] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, and et al. The open images dataset v4. *International Journal of Computer Vision*, Mar 2020.

[37] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: A backbone network for object detection, 2018.

[38] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2016.

[39] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2017.

[40] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014.

[41] Jiang-Jiang Liu, Qibin Hou, Ming-Ming Cheng, Jiashi Feng, and Jianmin Jiang. A simple pooling-based design for real-time salient object detection, 2019.

[42] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[43] A. Loquercio, A. I. Maqueda, C. R. del-Blanco, and D. Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 3(2):1088–1095, 2018.

[44] Jiyan Pan and Bo Hu. Robust occlusion handling in object tracking. 06 2007.

[45] L. Patino, T. Nawaz, T. Cane, and J. Ferryman. Pets 2017: Dataset and challenge. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2126–2132, 2017.

[46] B. Penin, P. R. Giordano, and F. Chaumette. Vision-based reactive planning for aggressive target tracking while avoiding collisions and occlusions. *IEEE Robotics and Automation Letters*, 3(4):3725–3732, 2018.

[47] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.

[48] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[49] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[50] Zhongzheng Ren, Zhiding Yu, Xiaodong Yang, Ming-Yu Liu, Yong Jae Lee, Alexander G. Schwing, and Jan Kautz. Instance-aware, context-focused, and memory-efficient weakly supervised object detection, 2020.

[51] Faegheh Sardari and Mohsen Ebrahimi Moghaddam. A hybrid occlusion free object tracking method using particle filter and modified galaxy based search meta-heuristic algorithm. *Applied Soft Computing*, 50, 11 2016.

[52] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection, 2019.

[53] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[54] Y. Song, K. Yoon, Y. Yoon, K. C. Yow, and M. Jeon. Online multi-object tracking with gmphd filter and occlusion group management. *IEEE Access*, 7:165103–165121, 2019.

[55] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[56] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection, 2019.

[57] Lachlan Tychsen-Smith and Lars Petersson. Denet: Scalable real-time object detection with directed sparse sampling, 2017.

[58] Jianfeng Wang, Rong Xiao, Yandong Guo, and Lei Zhang. Learning to count objects with few exemplar annotations, 2019.

[59] Zhongdao Wang, Liang Zheng, Yixuan Liu, and Shengjin Wang. Towards real-time multi-object tracking, 2019.

[60] Andreas Wedel, Uwe Franke, Jens Klappstein, Thomas Brox, and Daniel Cremers. Realtime depth estimation and obstacle detection from monocular video. volume 4174, pages 475–484, 09 2006.

[61] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric, 2017.

[62] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37:1–1, 09 2015.

[63] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[64] Y. Xiang, A. Alahi, and S. Savarese. Learning to track: Online multi-object tracking by decision making. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4705–4713, 2015.

[65] Z. Xu, Y. Ding, J. Shan, and X. Xie. An anti-occlusion correlation filtering tracking algorithm for uav. In *2018 IEEE International Conference on Progress in Informatics and Computing (PIC)*, pages 163–168, 2018.

[66] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Mueller, R. Manmatha, Mu Li, and Alexander Smola. Resnest: Split-attention networks, 2020.