Project Code: AAE035

THE HONG KONG POLYTECHNIC UNIVERSITY

BEng(Hons) in Aviation Engineering

2020/21

AAE4002 Capstone Project Final Report

Topic:

Vision-base of a Quadrotor UAV Navigation

LO, Li-Yu (17086854D)

YIU Chi Hao (17003167D)

Tang Yu (17086039D)

Report submitted in partial fulfilment of the requirements for the Honours Degree of Bachelor of Engineering in Aviation Engineering.

Date of Submission: 23/4/2021

Acknowledgement

We would like to express our sincere appreciation and indebtedness to the following significant project supervisors and teaching assistants to bring this project into fruition.

Prof. Chih-Yung WEN, Professor and Interim Head of AAE department, for his wholehearted support, words of encouragement and supervision throughout this project.

Dr. Boyang LI, for offering his valuable advices and comments upon accomplishment of this project.

Mr. Yu Rong FENG and **Dr. Sheng Yang CHEN**, for their knowledge, enthusiasm, and conscientious guidance during all stages of work.

Mr. Jeremy Chang, for his technical knowledge and kind assistance on UAV hardware setup.

Contents

Abstract

The ever-burgeoning growth of autonomous unmanned aerial vehicles (UAVs) has demonstrated a promising platform for utilization in real-world applications. Autonomous UAVs can be of invaluable assistance in both search & rescue (SAR) and reconnaissance missions if they are able to search and track target objects. Hence, this paper proposes a learning-based navigation system for quadrotor UAV that addresses 2 missions: to autonomously accomplish an object-searching task and an object-tracking task under an unfamiliar environment.

In specifics, we adopt deep-learning based YOLOv4-Tiny algorithm for the purpose of semantic object detection. We further consolidate the algorithm with a learning-based 3D object pose estimation method as well as a collision-avoidance path-planning solution in order to search and locate static target objects. And for object tracking, we fuse the object detector YOLOv4-Tiny with Kalman Filter to track the dynamic target object and design the back-end UAV actions to follow the target object.

We validate the presented system in both simulated situations and real-world environments. Experimental data are collected and analysed through the 'Gazebo' simulator program and several flight tests under the VICON arena.

The findings of the study demonstrate the effectiveness and reliability of such autonomous navigation system: in which the object searching system is capable to precisely search and locate target in an unknown environment with a collision-free flight. Meanwhile, the object tracking system is able to consistently track and follow the movement of the target object.

Keywords: *UAV; autonomous object-searching; autonomous object-tracking*

Abbreviations

The following abbreviations are frequently used in this report:

AP:	Average Precision
CG:	Centre of Gravity
CNN:	Convolution Neural Network
DNN:	Deep Neural Network
EDP:	Edge Detection Planner
EKF:	Extended Kalman filter
FoV:	Field of view
FPS:	Frames per second
FSM:	Finite State Machine
GNSS:	Global Navigation Satellite System
GPU:	Graphics Processing Unit
IIR:	Infinite Impulse Response
IOU:	Intersection of Union
KF:	Kalman filter
mAP:	Mean Average Precision
ML:	Machine Learning
NP:	Non-deterministic Polynomial-time
OpenCV:	Open-Source Computer Vison Library
OS:	Object Searching
OT:	Object Tracking
QP:	Quadratic Programming
RGB-D:	Red, Green, Blue – Depth
ROI:	Region of Interest
ROS:	Robot Operating System
SAR:	Search and Rescue
SILT:	Software-in-the-loop
TX2:	NVIDIA®Jetson™ TX2
UAV:	Unmanned Aerial Vehicle
UKF:	Unscented Kalman filter
YOLO:	You Only Look Once

1. Introduction

Unmanned aerial vehicle (UAVs) has revealed its unprecedented potential for both commercial and civil-government utilization in wide range of applications like urban area surveillance (Semsch et al., 2009), aerial photography (Gurtner et al., 2009), logistics (Škrinjar et al., 2018), and so forth. Specifically, vision-based navigation system that employs optical sensors has become a popular trend in UAVs application. It is widely known that traditional sensors like GPS degrades its accuracy due to the limited number of satellites (Araar & Aouf, 2014) and the signal interference in indoor environment (Dedes & Dempster, 2005; Koyuncu & Yang, 2010). Other sensors that are commonly used by UAVs, including lidar and lasers, also lead to shortcomings of weight and power consumption (Heritage & Large, 2009; Wallace et al., 2012), and therefore visual sensors become substantial alternative for UAVs. Study of Lu et al. (2018) pointed out the benefit of visual sensors compared to the conventional sensing technology in providing abundant real-time visual information of surrounding environment with remarkable anti-interference ability.

Vision-based UAV navigation system is exclusively advantageous for tasks that require distinct visualization and robust perception, for example, search & rescue (SAR) operation and reconnaissance application. The conventional SAR mission is time-consuming, labour-intensive, and hazardous for rescuers to physically ascertain the position of survivors in the aftermath of catastrophe. Meanwhile, reconnaissance work is tedious and inefficient. The employment of a vision-based quadrotor UAV is particularly a promising platform in assisting SAR and reconnaissance missions by virtue of its agile manoeuvrability to approach confined areas of low accessibility and its visual functionality to provide remote data in real time. On behalf of mankind, UAV is extremely helpful to search and locate the victims trapped in an indoor environment. Nevertheless, such searching operation of UAV is insufficient for mobile objects as the victims might move in real-world scenarios. Additionally, a reconnaissance mission may also entail continuous tracking of the target object. Therefore, apart from searching static objects, UAV should be equipped with the capability to track and follow dynamic objects. Both searching and tracking actions are critical as preliminary procedures for a typical SAR or reconnaissance mission.

The conventional methods used by UAVs in conducting the SAR or reconnaissance mission might not be competent enough to perform real-time searching and tracking. For instance, the

standard and outdated techniques employed the color and thermal imagery processing. Rudol and Doherty (2008) suggested a vision-based searching method with infra-red cameras while Sun et al. (2016) proposed a camera-based target identification system using color signatures. However, they all relied on image post-processing procedure on ground station to recognize objects. Alternatively, multiple sensor data fusion (Carrillo et al., 2012) in a single vision-based navigation system was chosen to accomplish object searching and tracking. Tomic et al. (2012) introduced a combination of laser and stereo vision odometry. Similarly, Mittal et al. (2019) integrated ground penetrating radars like bio-radars with stereo camera and GPS to detect victims in SAR missions. However, the system configuration with multiple sensors was expected to increase the payload, computational resources requirement, and battery power consumption of UAV significantly; besides, the detection method mentioned above was limited to human being only, indicated that the UAV system is not flexible enough to search other categories of objects. In addition, works of Burke and Murphy (2004), Yeong et al. (2015), Grogan et al. (2018) have proved that such usage of the UAV would be costly and arduous if conducting in manual teleoperation, which means that an autonomous UAV system is a fundamental requirement.

In recent years, benefiting from the rapid advancement of deep-learning-based technique, the deep-learning based system for small UAV like quadrotor is a prosperous area of research (Szeliski, 2010). Therefore, considering the aforementioned factors, we are motivated to develop a learning-based navigation system for quadrotor UAV to autonomously perform object searching and object tracking in an unknown environment. The system focuses on addressing 2 different problems: an autonomous object searching task that deals with static objects, and an autonomous object tracking task that focuses on a dynamic target object. Without the prior information of the environment, the proposed UAV systems are designed with deep-learning based perception to perceive surrounding environment and path-planning capability to process navigational decisions onboard during autonomous flights.

Our contributions (objectives) of work in this project are summarized as follows:

1. An autonomous object searching system for UAV is proposed. In which,
    a. the learning-based object detection algorithm and 3D object pose estimation method are extended and implemented in a novel system to search and to locate target objects.

b. A path planning solution with image-processing techniques (Edge Detection Planner) and minimum jerk trajectory generation is designed to avoid column-wise obstacles while searching objects.

2. An autonomous object tracking UAV system is proposed. In which,
    a. the learning-based object detector with Kalman filtering estimation is utilized for recognizing the target object and resolving the occlusion problem in tracking.
    b. a back-end path planning method with learning-based 3D object state estimation method for UAV to track and follow the object is presented in this study.

This paper is written in the style of academic paper despite the fact that it is a report for AAE4002 final year project. All writings are delivered in a concise and precise manner. The following content of the paper is organized as follows. Section 2 introduces the relevant literatures and Section 3 describes the details of the overall system architecture. Section 4 and Section 5 explain the methodologies of our works, specifically object-searching and object tracking. Section 6 presents the experiments, results, and analysis of result. Lastly, this paper is concluded with several recommendations for future research. Other supplementary materials like our implementation codes and video footage of experiments are attached in Appendix I.

2. Literature Review

This section covers some essential research of the related topics to provide readers general background information. It mainly comprises of object detection, object tracking, and UAV path-planning and trajectory generation module.

2.1 Object detection in UAVs application

Researchers have been attempting numerous approaches to achieve object detection through aerial imagery taken by vision-based UAVs over the last decades. As stated by Szegedy et al. (2013), Huang et al. (2014), and Russakovsky et al. (2015), object detection on image is typically defined as finding the position of objects (object localization) and classifying the corresponding categories of detected objects (object classification).

The research community led the trends in embedding object detection algorithms in the field of UAVs. Hulens and Goedemé (2017) applied conventional object detection method, particularly the deformable part-based model (Felzenszwalb et al., 2009), on UAVs for recording video of movie actors. Zhiqiang and Jun (2017) described the traditional object detector processed pipeline of 4 stages (Fig. 2.1) including multi-scale sliding window, hand-crafted features extraction, classification done by Supported Vector Machine (SVM) or AdaBoost classifier, and lastly Non-Maximum Suppression (NMS) and combined Bounding box for optimization of object detection performance. The aforementioned deformable part-based model was one kind of the Histograms of Oriented Gradients (HOG) detectors. However, this traditional approach encountered difficulties such as low capacity of data model, low robustness for various geometric or photometric changes, excessive computation cost, high inaccuracy, huge semantic gap (Bengio, 2009; Dai & Yang, 2010; Zhao et al., 2019; Nixon & Aguado, 2019), which were all crucial qualities for real-time application.

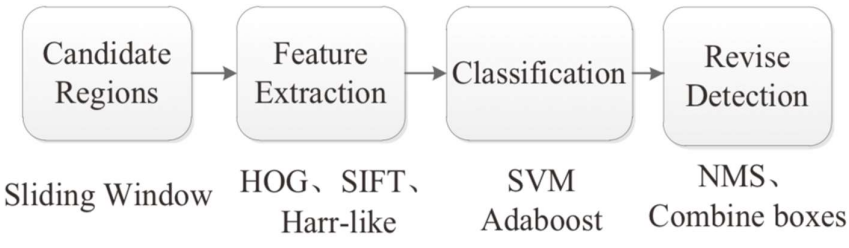


Fig. 2.1 Pipeline of traditional object detector by Zhiqiang & Jun (2017)

Other discrete object detection algorithms like point target detection algorithm (Westall et al., 2007) and generalized contour search algorithm (Goodrich et al., 2008) showed better performance than the preceding approach. However, they still suffered limitations of accuracy, speed, cost, and complexity. Until a later time, deep-learning-based approaches, especially the Deep Neural Networks (DNNs), have emerged as the key breakthrough for object detection in computer vision and UAVs industry. The state-of-the-art object detection algorithms, particularly the Convolutional Neural Networks (CNNs) series and 'You-Only-Look-Once' (YOLO) series, are both derived from the DNNs.

Andriluka et al. (2010), Bejiga et al. (2017), and Lygouras et al. (2019) fused the CNN-based algorithms with on-board visual sensors to achieve real-time object detection in conducting SAR missions. Meanwhile, Tijtgat et al. (2017), Kyrkou et al. (2018), and Feng et al. (2021) employed 'YOLO' series algorithm as the object detection framework for real-time UAVs application. Deep-learning based approach, both CNNs and 'YOLO' method, is deemed to be the powerful and prevailing object detector embedded in the vision-based navigation system, which also integrates well with the path planning problems of UAVs (Carrio et al., 2017).

2.1.1 CNNs for Object Detection

The Deep Neural Networks (DNNs), or Schmidhuber (2015) called it as "Deep Learning (DL) in Artificial Neural Networks (NNs)", is a substantial machine learning tool designed by Hinton and Salakhutdinov (2006) to reduce dimensionality of data. Through the idea of DNNs, Krizhevsky et al. (2017) published the revolutionary AlexNet Deep Convolutional Neural Networks (CNNs) model which was a supervised deep-learning algorithm for feature extraction. Goodfellow et al. (2016) elucidated the supervised learning algorithm learns how to output the target values by manually training a set of examples of inputs and desired outputs. In other words, CNNs object detector manages to identify the class of objects in inputted image provided people manually assign the labels of class to the corresponding objects within the training dataset.

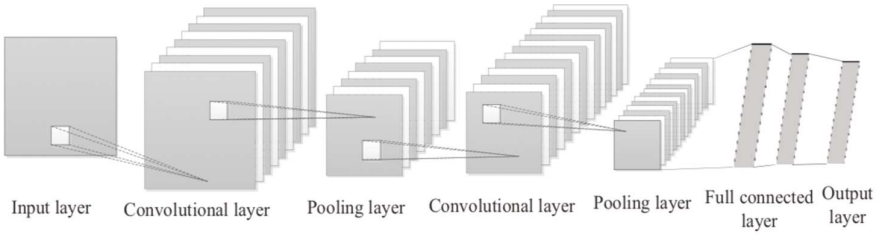


Fig. 2.1.1 CNNs architecture (Zhiqiang & Jun, 2017)

The architecture of typical CNNs is demonstrated in Fig. 2.1.1 above. The key structures are the subsequent convolutional layers with pooling layers for feature learning, followed by a fully connected layer and an output layer. The convolutional layer executes a "2D convolution of a 2-dimensional image I with a 2-dimensional kernel K" (Carrio et al., 2017), as shown in the equation below, to extract the features and to operate the nonlinear activation function like Rectified Linear Units (ReLU) or Sigmoid (Du, 2018). Then, the pooling function (e.g., average pooling, max pooling) in pooling layers maintains the scale invariance to small translations of the input and reduces the dimensionality to subsample of the data. Lastly, the linear classifier and fully connected neutrons in full connected layer use results of the convolution/pooling operation to classify the image into a specific label and further obtain one-dimensional feature vector.

$$C(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \qquad (2.1.1\ a)$$

The variant family of CNNs object detector, including R-CNN, Fast R-CNN, Faster RCNN, and R-FCN, are all region proposal-based detectors, or basically regard as two-stage detector. They all show respective approaches for improving the performance of object detection in three reasons: the accuracy, speed, and the computational complexity to implement in real-time platform. The following paragraph lists the predominant modifications:

(a) Region with CNN (R-CNN), proposed by Girshick (2014) and his team, implemented region proposals algorithm like 'selective search' algorithm (Uijlings et al., 2013) to extract 2000 category-dependent region proposals, then computed feature through pre-trained CNN model, and lastly classified them by linear Support-Vector-Machine

(SVM) classification. It achieved high detection accuracy but faced critical obstacles like high computationally inefficiency and losing of image information.

(b) Fast R-CNN (Girshick, 2015) substituted the final pooling layer by Spatial Pyramid Pooling (SPP)-layer (He et al., 2015), and applied Region of Interest (ROI) pooling and proposal reflection based over the SPP layer, and replaced SVM with Softmax classifier and Bounding-box method with linear regression layer. The computation time for training got slightly lesser but still not fast enough in real-time operation and embedded platform.

(c) Faster R-CNN (Ren et al., 2015), added 'Region Proposal Network' (RPN) on the feature maps instead of the 'selective search' algorithm to produce a set of high-quality object proposals. The detection accuracy was higher, but still computationally heavy due to its nature of two-stage processing pipeline.

2.1.2 YOLO for object detection

Instead of detecting objects in two-stages method, like 'RPN + classifier design' of Faster R-CNN, another type of object detector with DNN-based regression and anchor mechanism, is adopting one-stage method that performs object localization and object classification in single pipeline. The famous ones are 'You Only Look Once' (YOLO) series, RetinaNet, and Single Shot MultiBox Detector (SSD).

In specific, 'YOLO' (Redmon et al., 2016) took advantages from CNNs based architecture which it applied single CNN on the whole image, generating bounding boxes coordinates, confidence level, class probability in one evaluation. The working principle of 'YOLO' framework would be further clarified in the following content. Meanwhile, SSD (Liu et al., 2016) made use of different scale of grid cells and small convolutional filter to increase the accuracy. RetinaNet (Lin et al., 2017) focused on using focal loss function to balance the foreground-background class. In short, they all share the great advantages in providing real-time speed and high compatibility to embedded platform, but relatively low accuracy compared to the two-stage region-based object detectors (Huang et al., 2017).

According to Redmon et al. (2016), the input image is divided into $S \times S$ grid cells and predictions are made by B bounding boxes weighted by the confidence level and C number of

class probabilities, as demonstrated in Fig. 2.1.2. There would be only one set of class probabilities per grid cell, as shown in class probability map. The prediction result is calculated and encoded as followed equation:

$$S \times S \times (B * 5 + C) \qquad (2.1.2\ a)$$

In particular, confidence level indicates the confidence of model that an object exists in bounding box. There would be one confidence score for individual bounding box and the score is defined as:

$$Confidence\ score\ (Conf) = P_r(Object) \times\ IOU_{pred}^{truth} \qquad (2.1.2\ b)$$

- $P_r(Object)$ is the probability of objects in the bounding box, it would be 0 for no object, 1 for object contained.
- IOU (Intersection Over Union), a fraction between 0 and 1. Huang et al. (2018) defined IOU as "the overlapping area between the predicted bounding box and ground truth, and union is the total area between both predicted and ground truth". The ground truth is obtained by developers during the training process in custom datasets.
- When the object is contained within the predicted box, the ideal *Conf* equals *IoU*, otherwise, *Conf* = 0.
- The bounding box that is predicted to potentially contain object would be keep if its confidence score is higher than the manually settled confidence threshold value. Any other remaining predicted bounding boxes are disregarded if the confidence score is lower than the threshold value, normally 0.5.

Each bounding box consists of 5 predictions: bounding box center coordinate *x*, bounding box center coordinate *y*, width *w* relative to the whole image, height *h* relative to whole image, and lastly the confidence score. For example, for *i*th object, the output vector of the bounding box would be $\{x_i\ \ y_i\ \ w_i\ h_i\ \ conf_i\}$. This is also the reason of $(B\ *\ 5)$.

By multiplying the confidence score with the *C* conditional class probabilities, we obtain the class-specific confidence scores for each bounding box:

$$P_r(Object) \times\ IOU_{pred}^{truth} \times\ P_r(Class_i|Object) =\ P_r(Class_i)\ \times\ IOU_{pred}^{truth} \qquad (2.1.2\ c)$$

- The C conditional class probabilities is presented by $P_r(Class_i|Object)$
- The output on the right hand side is the confidence score for specific class in each bounding box, which is usually the confidence score shown during the real-time detection.

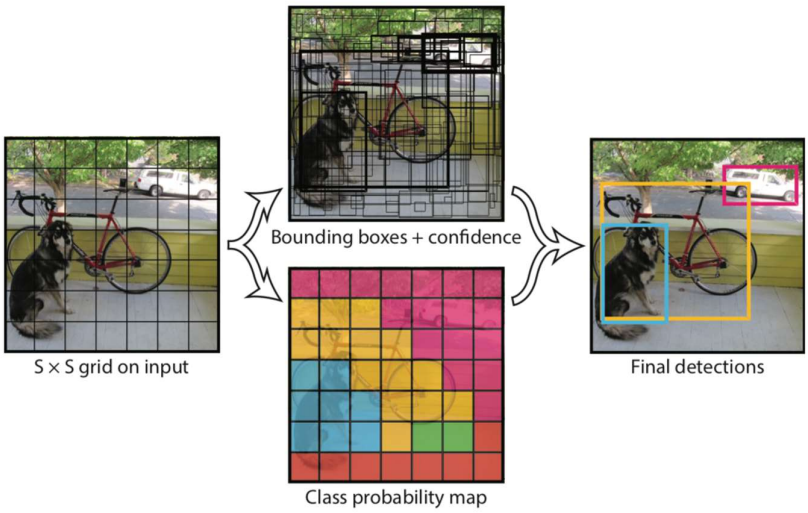


Fig. 2.1.2 Single pipeline of 'YOLO' object detection (Redmon et al., 2016)

The training process for this supervised deep-learning-based 'YOLO' requires standard data augmentation methods (e.g., random scaling, adjustment of image orientation and parameters), optimization of sum-squared error loss and multi-part loss function, labelled data from public datasets or in-house datasets.

'YOLO' achieved real-time detection speed, but occurred inaccuracy in object localization, especially for small objects or adjacent objects in image. Therefore, several updated versions of 'YOLO' were developed to improve speed, accuracy, and availability for embedded computing devices with limited computational resources. The key advancements are summarized as follows:

- YOLOv2 (Redmon and Farhadi, 2017) changed the 'YOLO' framework by batch normalization, high resolution classification, anchor boxes prediction, k-means clustering, elimination of fully connected layer, and the invention of DarkNet-19 network.
- YOLOv3 (Redmon and Farhadi, 2018) used feature pyramid network, logistic regression, multi-scale features, independent logistic classifier, binary cross-entropy loss, and DarkNet-53 as the backbone of training model. DarkNet-53 was able to obtain the highest billion floating point operations per second (BFLOP/s) among the backbone networks, indicated that its high compatibility with GPUs. Lin et al. (2017) pointed out the YOLOv3 outperformed other object detectors with lesser inference time and higher mean Average Precision (mAP).
- YOLOv4 (Bochkovskiy et al., 2020) adopted CSPDarknet53 as backbone, PANet path-aggregation network as neck, and YOLOv3 as the head architecture with new data augmentation mosaic and self-adversarial training to obtain unprecedented performance in terms of accuracy and speed.

In fact, such 'YOLO' object detection systems have high computational requirement on the embedded devices, which powerful graphics processing unit (GPU) is a fundamental component (Carrio et al., 2017). The computational resource on UAV's GPU remains the most difficult issue that causes slow speed and constrain the usage of state-of-the-art object detectors. Hence, Shafiee et al. (2017) proposed a 'Fast YOLO' framework to speed up the object detection by 3.3 times. Besides, Huang et al. (2018) recommended the 'YOLO-LITE' that works well with non-GPU computer.

Lastly, 'YOLO v2-Tiny' and 'YOLO v3-Tiny' created by Redmon (2016) and 'YOLO v4-Tiny' by Bochkovskiy (2020), significantly reduce the network complexity of original 'YOLO' framework. 'YOLO v3-Tiny' (Adarsh et al., 2020) delivers higher attainable frame per second (FPS) and lower network size than 'YOLO' model. Mao et al. (2019) commented that these light weighted 'YOLO Tiny' versions are all intended to reduce the model size and floating-point operations (FLOPs) and network parameters in order to realize real-time performance on low-power embedded GPUs. Zhao et al. (2020) agreed that 'YOLO Tiny' frameworks made tradeoff between accuracy and detection speed, with the former one being apparently degraded yet decent.

In summary, this project tends to relate the latest 'YOLOv4-Tiny' on our UAV's resource-limited onboard computer. We consider the high detection speed (high FPS) with acceptable precision and portability of C-based release to be an adequate and suitable solution for our objectives. The details of implementation of 'YOLO v4-Tiny' is further elaborated in the methodology Section 4.

2.2 Object tracking in UAVs application

Built on the basis of object detection and beyond it, remarkable development of computer vision technologies in recent years encourages a new paradigm for object tracking in UAVs' application. In addition to localizing and classifying the target object, object tracking involves the motion estimation or trajectory prediction of objects across a sequence of frames (García-Rodríguez et al., 2009; Li et al., 2013; Smeulders et al., 2013). Ciaparrone et al. (2020) described that the majority of the state-of-the-art object tracking frameworks comprise steps of (1) object detection algorithms, (2) motion prediction methods, and (3) association problem to assign the numerical identification to each object. As for multiple object tracking, affinity computation and data association are the most important factors to ensure a continual tracking process on particular object.

It is much more challenging than object detection as it faces uncertainties and complexities in aspects including scene illumination changes, abrupt motion of objects, occlusions, noise corruption in images, camera motion blur problem, and so forth (Lu et al., 2018). Specifically, Lee et al. (2014) pointed out that occlusion (i.e., the tracked object is not available for camera to keep monitoring its motion state while the object is still present at the same scene) is the most common issue that happens in object tracking, regardless of its type (e.g., partial occlusion, full occlusion, inter-object occlusion), which leads to defects of tracking loss and identity switches. To handle occlusion problem, fusion methods with linear dynamic models or nonlinear dynamics models are recommended by Lee et al. (2014).

Nevertheless, object tracking in image planes of camera sensor and object tracking in vision-based UAV system are two different fields of study as the latter requires additional relative-control and coordination of UAV in autopilot flight (Wise & Rysdyk, 2006). The fast movement of UAV on air, field of view (FoV) of onboard camera, and backend actions of UAV to maintain visible distance with goal objects are all important considerations in planning

object tracking with UAVs. For a deeper understanding of object tracking topic, some fundamental literatures of object tracking models in image plane are reviewed before discussing the implementation results in UAVs application.

There were comprehensive literature surveys written by Pulford (2005) and Yilmaz et al. (2006), which provided detailed descriptions of representative approaches in the early development of object tracking. In the past, some researchers suggested 'point tracking' methods with data associations like Joint Probabilistic Data Association Filters (JPDAF) (Rasmussen & Hager, 2001) and Probabilistic Multiple Hypotheses Tracking (PMHT) (Streit & Luginbuhl, 1994), which were based on Bayesian probability (Tao et al., 2002). 'Kernel tracking' methods that tracked the shape and appearance of objects, for example, Kanade–Lucas–Tomasi (KLT) (Shi, 1994) and Support Vector Tracking (SVT) (Avidan, 2004), relied on the feature extraction to memorize the appearance of objects. Silhouette or contour tracking was another type of approaches which tracked contour information encoded in non-rigid objects (Bertalmio et al., 2000; Nikolaidis et al., 2009; Naveenkumar et al., 2018). All these traditional methods, especially contour-based, could not address the critical drawbacks, including occlusions, camera visual shelters from key features (Chan & Vese, 2001), rapid motion of objects (Hu et al., 2012), background confusion, and computational cost.

In last decade, the research community initiated the object tracking by 'tracking-by-detection' algorithm (Breitenstein et al., 2009), in which the generative, discriminative, and hybrid statistical modeling were fused to improve the performance of object tracking (Li et al., 2013). Nowadays, benefitting from the revolutionary enhancement of deep-learning, different deep-learning tracking frameworks were presented such as unsupervised deep-learning algorithm (Wang and Yeung, 2013), pre-training network combined with correlation filter (Danelljan et al., 2015), siamese-based tracking network (Tian et al., 2020) as well as the spatially supervised recurrent convolutional neural networks with YOLO and Long Short-Term Memory (LSTM) (Ning et al., 2017).

Another noteworthy development was the 'DeepSort' tracker proposed by Wojke et al. (2017), whose idea was inspired by 'Simple Online and Realtime Tracking' (SORT) (Bewley et al., 2016). Specifically, 'DeepSort' applied CNN to the SORT framework that implemented Kalman filtering in image space and Hungarian method. It learned the features of tracked object and predicted the future associated trajectories and positions of the objects of interest. The

recent paper of Punn et al. (2020) demonstrated the positive results of using YOLOv3 with DeepSort tracking scheme to observe the social distancing situation. Both Kalman filter and Hungarian helped motion prediction and association of object tracking. The study of Chen (2011) proved the effectiveness of Kalman Filter (KF) in vision-based robotic application and Huang et al. (2008) further verified the power of Hungarian method in handling data association procedure in robust object tracking. The details of Kalman Filter (KF) would be clarified in Section 5.1.2 as our paper endorses the Kalman filter in resolving problems of occlusion when predicting motion.

Speaking of object tracking with UAVs, in 2005, Ryan and Hedrick (2005) used UAVs installed with infrared camera sensors to track the helicopter during SAR missions, and Kalman Filter estimation was proved as a great solution to approximately predict helicopter's position and velocity. Rathinam et al. (2007) tried vision-based following systems of ground vehicles on UAVs to autonomously track the path of river or coast; yet they failed at high error rate and low robustness. The better approaches in this field were the image feature processing with Kalman filtering (Siam & ElHelw, 2012) and the appearance-based tracking algorithm on color and depth data (Al-Kaff et al., 2019). Lastly, Xu et al. (2018) made a good paradigm in employing YOLO and JPDA on small-scale UAVs to achieve real-time multiple object tracking.

In short, deep-learning-based object detector with filter algorithm is considered as a novel and promising approach with high flexibility in categories of target objects, reduced occlusions, and real-time processing speed. As for this project, we relate our work with the utilization of YOLO object detector and Kalman Filter to perform single object tracking. The details are incorporated in Section 5.

2.3 UAV Path Planning and Trajectory Generation

2.3.1 Path Planning

Path-planning is a necessary process for UAVs to generate safe flight paths to the points of interest, overcoming both physical and operational constraints. Compared with 2-D scenario, 3-D path planning faces much more dynamic constraints and kinematic constraints, and there is no universal solutions as 3-D path planning is a NP-hard problem in terms of optimization theory (Yang et al., 2014). Traditional path-planning algorithms could be classified into five types as proposed by researchers Yang et al. (2014):

(1) sampling search algorithms (e.g., Rapidly-exploring Random Tree (Cao et al., 2019) Probabilistic Roadmap (Yan et al., 2013), artificial potential field (Radmanesh et al., 2018))
(2) node-based optimal search algorithms (e.g., Dijkstra's algorithm (Johnson, 1973), A* (Hart, 1968), D* (Carsten et al., 2006))
(3) bio-inspired algorithms (e.g., genetic algorithm (Pehlivanoglu et al., 2007), particle swarm optimization (Foo et al., 2009))
(4) mathematic model-based algorithms (e.g., optimal control (Miller et al., 2011))
(5) multi fusion-based algorithms (i.e., a combination of several algorithms)

Among all traditional algorithms in these five categories, some were developed to perform path-planning based on known map of the surrounding, while others focused on dynamic route planning in an environment where threats were unknown. For instance, algorithms like A*, Probabilistic Roadmap, Dijkstra's algorithm generated a path based on global knowledge of known obstacles, while other dynamic path-planning algorithms, including D*, particle swarm optimization, bug algorithm series (McGuire et al., 2019), artificial potential field algorithm, functioned with respect to unknown threats to perform local path planning (i.e., obstacle avoidance). The following content briefly introduces the aforementioned five categories of path-planning.

(1) Sampling based algorithms will sample a space into a series of nodes. One category of sampling based algorithms, take RRT as a representative, could independently output a path. Some improvements to RRT, such as two-tree roots or replanning function, were introduced to form the RRT Connect and RRT*, respectively (Kuffner et al., 2000) (Chao et al., 2018). In addition, Artificial potential field method, which also generated a path solution independently, could lessen the computational burden, but was vulnerable to local minima trap issue (Radmanesh et al., 2018). The other category, such as PRM, generated a set of nodes and then converted into a node-based optimal searching problem.

(2) Node-based optimal search algorithms, such as, Dijkstra's algorithm and A*, were widely employed in the industry. A* algorithm could output an optimal path faster than Dijkstra's algorithm by virtue of a heuristic function. Both A* and Dijkstra's algorithm

was limited to a condition where the global information is already sensed and constructed. D* algorithm, an extension of A* algorithm, broadened its application to unknown environment, but the limitations were some inessential turnings as well as the assumption of constant transition cost from a specific grid to its neighbours (Ferguson & Stentz, 2007).

(3) Genetic algorithm, one popular kind of bio-inspired algorithms, employed the principle of evolution. This evolutionary algorithm searched paths from multiple points at a time and then optimized a path based on an objective function, which evaluated the fitness of each candidate. Subsequently, an iterative process was initiated whereby a set of individuals become parents, and the mutation and crossover procedures started off by generating next populations until an optimal solution was reached (Yang et al., 2014). Bio-inspired algorithms were based on heuristics, and were robust searching algorithms to cope with complex, unstructured constraints; yet bio-inspired algorithms generally imposed long iteration time premature and suffered the problem of premature convergence (Radmanesh et al., 2018; Pehlivanoglu et al., 2007).

(4) Mathematic model-based algorithm modelled all constraints in a workspace in mathematical forms accurately, which necessitated large computational power. For example, Miller et al. (2011) considered different threats into various mathematical models, formed a 2-D boundary value problem, and used the solution to obtain 3-D optimal path from an admissible path, during which both kinematic and dynamic constraints were taken into account and a cost function was bounded.

(5) Multi-fusion based algorithms was a term coined by Yang et al. (2014), which indicated an integration of different algorithms to achieve optimal, time-efficient, power-saving path-planning in a 3D space. For instance, PRM integrated with A* to output an optimal path (Yan et al., 2013); artificial potential field algorithm was integrated with a visibility graph and Voronoi diagram to tackle the local minima trap issue (Masehian et al., 2004).

In our project, we consider that the UAV has no pre-defined knowledge about the environment that it will traverse, which we are lack of the global information of the mapping or obstacles. Hence, a dynamic route search algorithm regarding local path planning is more applicable to

our interest as well as the SAR scenarios. A deeper research on local path-planning is provided in following part.

Local path planning algorithms, including bug series, artificial potential field, D* algorithm, could be defined as real time obstacle avoidance methods that employ sensory information to generate contingency route plans (Vckay et al., 2017). For example, in local path planning bug 1 and bug 2 algorithms, the robot will initially follow a straight line that connects the starting point to the goal. Once detecting an obstacle, it travels along the obstacle boundary, during which the distance from each point to the final goal will be recorded and, if necessary, updated before a leaving point is found to complete the local path planning (Lumelsky & Stepanov, 1987). In artificial potential field method (Radmanesh et al., 2018), the influence of goal point is modelled as attractive potential while the obstacles are regarded as repulsive potential. The UAV, modelled as a point aiming to reach the minimum potential, will avoid any unknown obstacles because it will repel the repulsive force and navigate towards the lowest point that is the goal point. Some commonly-used attractive potential functions are Conial Potential, Quadratic Potential, Combined Potential, and a gradient descent method can be employed to obtain the lowest total potential (Choset, 2010). Besides, D* algorithm will do back-searching from the goal to the starting point, and has a "Modify-Cost" function to modify the cost once an obstacle is encountered. Subsequently, D* will form a temporal map nearby the obstacle and initiate path-replanning (Stentz, 1997). To conclude, normally, an autonomous vehicle will follow a pre-set straight line from the initial point to the goal until it senses obstacles. Subsequently, a local planning will be initiated, in which the vehicle deviates from the pre-set line and simultaneously updates useful information such as the current position and the obstacle leaving point.

2.3.2 Trajectory Generation

Trajectory generations works by taking a solution obtained by a path-planning algorithm and outputting a smooth and continuous trajectory curve considering UAV system's differential constraints and time parameter. Millinger & Kumar (2011) proposed a minimum snap trajectory generation algorithm, which represented the quadrotor trajectories into piecewise polynomial functions and then outputted an optimal solution by solving a quadratic programming (QP) problem. Specifically, the algorithm figured out the parameters of the trajectory polynomial, which formulated the trajectory into a constrained optimization problem and aimed to minimize the snap (i.e., the third derivative of velocity). After considering

equality constraints (e.g., velocity, acceleration) and inequality constraints (e.g., corridor constraints), the multivariate quadratic function was optimized by leveraging a QP solver. Richter et al. (2016) presented a method that obtains minimum snap trajectories in closed form, using an unconstrained quadratic program which enhances computation efficiency and ensures the safety of trajectories in the case of high-order polynomials and numerous segments by iteratively adding in-between waypoints. In addition to the minimum snap trajectory generation, minimum jerk (i.e., the second derivative of velocity) trajectory generation is also widely employed, such as the usage for visual-inertial navigation (Ryll et al., 2019) and space manipulator (Huang et al., 2006). Through minimizing the jerk, the vehicle vibrations can be mitigated, and the vehicle control efficiency and stabilization can be enhanced (Huang et al., 2006).

In sum, combining the understanding of path-planning problem and trajectory generation, we relate the implementation of local path-planning and optimized trajectory with minimum jerk trajectory generation method in our object searching task to ensure a smooth trajectory and great stability during the autonomous flight. The details are explained in Section 4.3. Furthermore, as for the object tracking, aggressive flight in path-planning is excluded, and a safe yet relatively simple path planning is preferred (Section 5.2).

3. System Overview

This section briefly describes the hardware platform that we used to conduct the experiments and the overall software architecture of our designed UAV system. The main component of our vision-based system is an Intel RealSense D435i stereo camera (Fig. 3(a)) for visual sensing and depth acquisition as it is proved to have light weight, wide field of view (FoV) with global shutter for moving camera motion (Ahn et al., 2019), high depth accuracy and stability (Carfagni et al., 2019). Besides, we employed a powerful GPU, Jetson TX2 onboard computer (Fig. 3(b)) to process deep learning-based algorithm on our small-scale quadrotor UAV platform. The deployed flight controller is Pixhawk PX4 (Fig. 3(c)) and an external 14-camera VICON Mocap system is utilized for map-less indoor visual localization purpose (Kushleyev et al., 2013; Zou et al., 2016). As evaluated by Merriaux et al. (2017), such platform is extraordinarily helpful and performs outstandingly in providing altitude and position feedback to UAV system in the experimental stage. In correspondence to the engaged VICON system, three spherical markers are attached to the airframe of quadrotor UAV. Moreover, our framework is supported by the Robot Operation System (ROS), whose *MAVROS* package is used to communicate PX4 flight controller and planner node at onboard computer, while *VICON package* ('vrpn_client') is used to connect VICON server and UAV onboard processor. Apart from VICON arena, Gazebo simulator and software-in-the-loop (SILT) program is chosen to conduct virtual experiments and simulation.

Fig. 3(a) Intel RealSense D435i stereo camera and Fig. 3(b) Jetson TX2 onboard computer

Fig. 3(c) PX4 controller

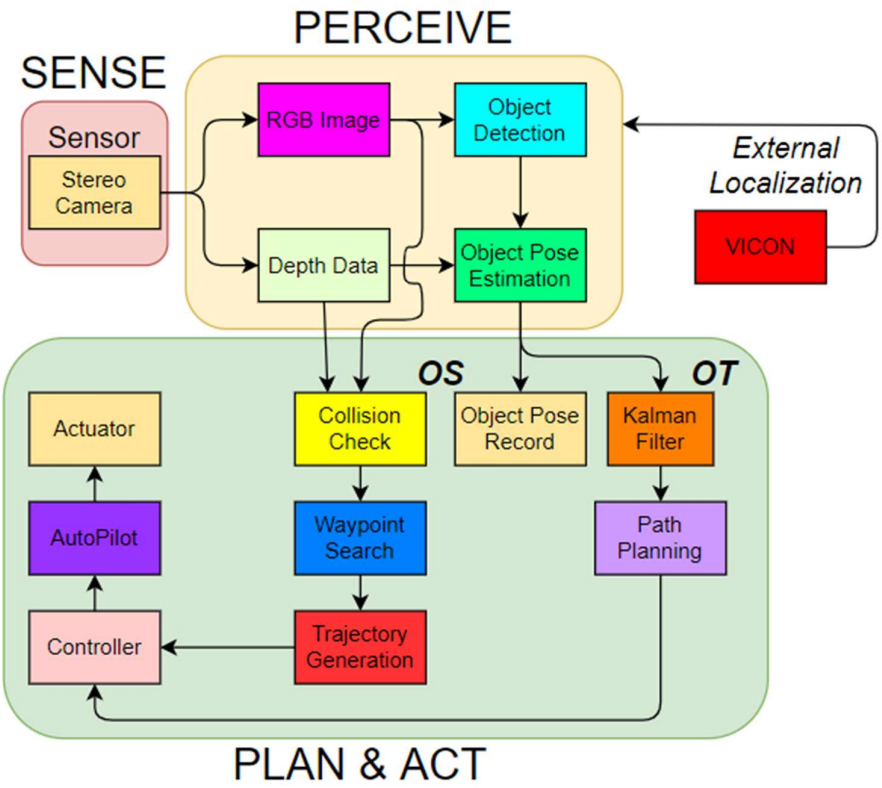


Fig. 3(d) Software architecture of our vision-based navigation system

The predominant software architecture of our autonomous vision-based system consists of the (1) front-end sensing, (2) perceiving module, and (3) back-end path planning and acting work, and (4) external localization. It is common to include localization and mapping area in UAV

field of study. Yet, in this study, the emphasis is to develop perception ability with deep-learning based methods, and further coordinate UAV to perform corresponding path planning and actions. The problems of the localization and mapping could be temporarily disregarded by deploying an external localization system, VICON motion capture system, to support the entire UAV system. In this integrated system, the components could be separate into 2 sub-streams, which are the Object Searching (OS) and Object Tracking (OT), because these two functions are supported by different approaches.

Object Searching (OS)

To commence with the sensor information achieved by the onboard stereo camera, the UAV system perceives the RGB image and the depth data. By the deployment of deep-learning based object detector YOLOv4-Tiny, the UAV system is able to detect objects in its field of view (FoV). The generated 2D bounding boxes fused with the depth measurement from camera and the IIR filter-based methods to obtain the 3D pose estimation of objects. Meanwhile, the depth data is integrated with image-processing technique, particularly Edge-Detection Planner, to perform obstacle avoidance and to generate flight trajectory with possible waypoint. The trajectory generation is further optimized with 'minimum jerk' method before the control stage of autopilot. A Finite State Machine (FSM) model for our quadrotor system is included in such OS system. The OS system is explicitly explained in step-by-step guide in Section 4.

Object Tracking (OT)

The OT system mainly comprises of important components like object detection algorithms, 3D object pose estimation, Kalman Filter (KF) prediction module, and the back-end decision-making and path planning of the UAV system. In brief, the coordinates of 2D bounding box produced by object detector YOLOv4-Tiny fused with the 3D object pose estimation to provide UAV system the 3D position of the tracked object. Kalman Filter module is integrated to aid in object motion prediction and lastly a path planning components with Finite State Machine (FSM) model to perform safe tracking flight. The detailed information of OT system is precisely described in Section 5.

4. Methodology – Object Searching

To use an UAV to search a goal object, the UAV system needs to own abilities to locate and to classify the objects in its view while flying in a collision-free way. Hence, the systematic approaches of object searching module in our project mainly consist of two parts: an object positioning function based on 'YOLOv4-Tiny' deep-learning based algorithm and a local path planner which utilizes image processing techniques. The system is designed to be capable of entering an unknown environment without colliding with obstacles and searching for the target object. This section aims to present the methodology of our object searching module. In particular, Section 4.1 and 4.2 will discuss how the UAV locates the target object, whereas section 4.3 focuses on the obstacle avoidance method. As for section 4.4, the Finite State Machine utilized in the module will also be covered.

4.1 Object Detection (object localization and classification)

To perform searching task on target object, the foremost and essential procedure would be the 2-D object detection on real-time aerial images. We have applied the state-of-the-art YOLOv4-Tiny algorithm as our object detection solution due to the needs of robustness, high detection speed, and low computational cost during the searching process. The following content of Section 4.1 mainly discusses the works of implementing YOLOv4-Tiny in our UAV navigation system, particularly the training process in in-house dataset and the 2D bounding box prediction.

4.1.1 Dataset Establishment and Training Process

4.1.1 (a) Dataset collection

The first step in using the open-source 'YOLOv4-Tiny' would be the preparation of customized dataset for training. According to Bochkovskiy (2019), in order to improve the detection performance of the training model, the custom dataset should contain data images with random and dissimilar illumination conditions, scales, view of angles, aspect ratios, resolutions, backgrounds, and so on, while it is considered that each training class should have at least 2000 images. Meanwhile, in order to avoid overfitting issue and to improve training result, it is suggested to have a validation dataset to provide an unbiased assessment of a model fit on the training dataset (Brownlee, 2017). Ripley (2007) defined the purpose of validation set to tune the parameters of detector. Hence, the entire dataset would comprise subsets of training set and validation set.

Our model aims to detect three classes of objects (i.e., winnie-the-pooh doll, yellow bulb ball, human), whereby the pooh and yellow bulb ball are designed as the desired target object of our project, while the human class is considered as the potential target object in SAR application or reconnaissance. Therefore, we established our in-house dataset, including 13,500 images in total, which was composed of 2000 training images plus 500 validation images (4:1 ratio) for each class, as well as 6000 background images with no target object. The 6000 background images are designed as negative images to further raise the accuracy of model, because it will learn to detect no object in a scene, and thus reduce the false positives results (Bochkovskiy, 2019). We carried out the observations of Feng et al. (2021) in preparing aerial scale images which approximately accord with the captured view of flying UAV. Additionally, within the training images, there were many images that contain multiple objects (i.e., pooh, yellow bulb ball, human) in a single frame, which would enhance the detection accuracy in the scenario that multiple target objects appear in the same scene.

Some representative images of our dataset are displayed in Fig. 4.1.1 (a).

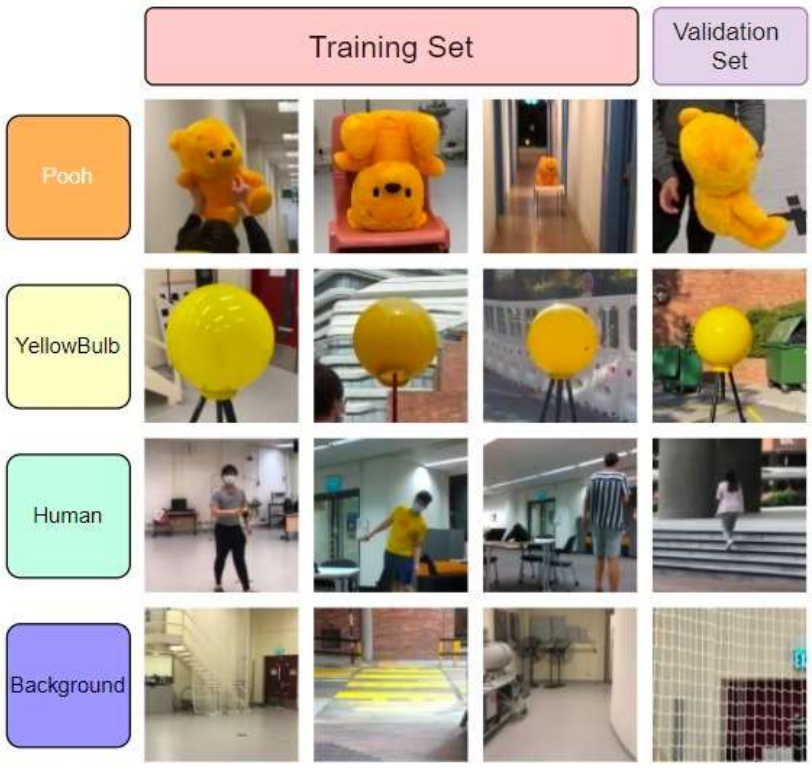


Fig. 4.1.1 (a) random image samples of the custom dataset

4.1.1 (b) Dataset labelling

Data labelling is an important process of indicating the interest of object as well as providing the ground truth bounding box of the image dataset to the computer, so that it could calculate the IOU (intersection of union) and the confidence score, and further develop the optimal weights for the model (Karpathyet al, 2017). As explained in the Section.2, such supervised deep-learning-based algorithm requires a set of examples of inputs and desired outputs.

Therefore, after the collection of data, we manually labelled our images by providing bounding boxes and the corresponding class names (i.e., Pooh, Yellow Bulb, Human). One important note is that when labelling an image that contains multiple objects, it is necessary to label all objects without exception such that the final detection performance can be guaranteed. Otherwise, if some of the objects are intentionally or unconsciously omitted, the final training result may be degraded. In addition, image dataset could also be obtained from public dataset, such as 'ImageNet', 'MS COCO', 'CIFAR -10', 'PASCAL VOC', 'Google's Open Images', which contain many common classes and can readily generate labels using OIDv4 toolkit. However, as we want to customize our training model to detect uncommon objects (e.g., Pooh), we eventually employed an annotation tool to manually label the images. The Fig.4.1.1 (b) below demonstrates the labelling procedure of our dataset.

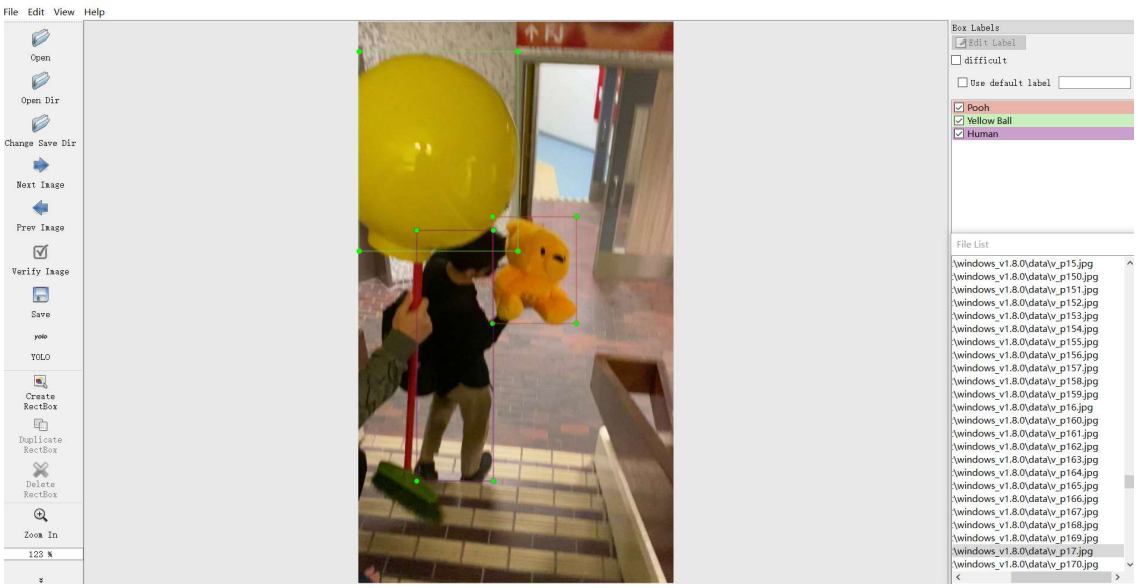


Fig. 4.1.1 (b) Manual labelling procedure

4.1.1 (c) Model Training

YOLO training could be operated both in local machine compiled with Darknet neutral network and in virtual machine 'Google Colab'. In our project, both methods were adopted to train Yolov4 model and Yolov4-tiny model. During training, specifically, we conducted multi-scale training in which different input resolutions of 320 × 320, 416 × 416, 512 × 512, 608 × 608 were examined in Yolov4-tiny to evaluate the best performance. The results are incorporated in Section 6.1.1.

The description of model training process in layman's term is the model generates a general predictor function to produce a predicted output, and the computer compares the predicted output with the actual output in repetition. Subsequently, through the continual comparison on large number of iterations, the discrepancy is calculated by loss function, and the result is referred as loss, or cost. Two important metrics that quantitatively measure the performance in the training process are the mean Average Precision (mAP) and the Loss. In short, the intentions of the training are to maximize the mAP and to minimize the loss.

(1) Mean Average Precision (mAP)

Mean Average Precision (mAP) is the mean value of Average Precision (AP) of each single class. The AP is defined as the area under the precision-recall curve (i.e., $AP = \int_0^1 p(r) * dr$), where the values of precision and recall both lie within 0 to 1. It represents the detection accuracy of target object in images.

To be more specific, Tayara and Chong (2018) clarified that:

$$precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \qquad (4.1.1\ a)$$

$$recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \qquad (4.1.1\ b)$$

- True Positives: a correct prediction of target object, IOU > 0.5
- False Positives: a wrong prediction as positive when there is no target object, IOU < 0.5

- False Negatives: a failure to predict a target object that was essentially there (Hui, 2018)

Empirically, a training process is deemed as effective if the mAP reaches acceptable value (e.g., 75% for YOLO-Tiny, 90% for YOLOv4) and levels off after a certain number of training epochs. As the iteration number of the model increases, the mAP also gradually increases because the model is capable to detect the target object more accurately after going through a large number of practices.

However, there exists a limit at which the increment of iterations would not raise the mAP anymore. This situation is called as 'overfitting', reflecting that the mAP reaches the peak after a certain number of iterations and then stagnates or even begins to decline. The overfitting issue then necessitates the usage of weights file at the early stopping pointwhichpoint (as depicted in Fig.4.1.1(c)), which could be found through checking the mAP of different previous weights. As depicted in Fig.4.1.1(c), the error of validation set increases after the iteration number passes the early stopping point, implying that the detector can only perfectly detect target objects from the training dataset, but would give poor performance when detecting test objects in real applications. This is the very reason that the training process should be terminated when the overfitting issue starts (Bochkovskiy, 2019; Lygouras et al., 2019). In simple words, the last saved weights file (training result) may not necessarily guarantee the best mAP of detection performance, and hence we need to check the iteration number and the weights file that gives the highest mAP.

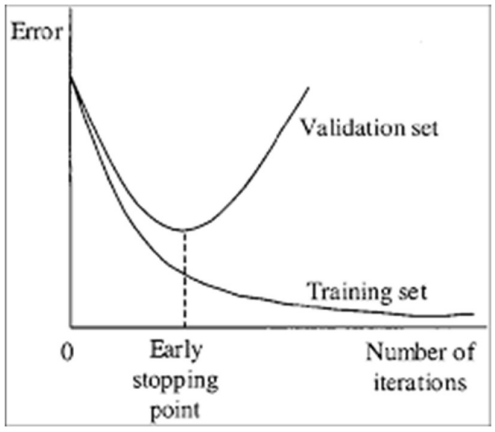


Fig.4.1.1(c). Illustration of the early stopping point during training iterations (Bochkovskiy, 2019)

(2) Loss

During the training process, the model is executing an optimization of sum-squared error loss and multi-part loss function to reduce the overall loss. In particular, the loss function consists of classification loss, localization loss, and confidence loss of the trained model. For the details of the complicated loss function, we refer the readers to the research of Redmon et al. (2016) and Jiang et al. (2020) as this project does not focus on developing the network of object detection framework. We trained the model until there was no significant drop of loss, which indicated that the discrepancies between our model predications and the ground truths were sufficiently low.

This paragraph specifically presents various details of our operating procedures throughout the training process for interests of readers. Several files are required to be carefully configured before a successful training could be achieved. Those were: *cfg* files, *obj.names* file, *obj.data* file, *train.txt* file, *test.txt* file. Our *cfg files* were edited based on *yolov4-obj.cfg* files and *yolov4-tiny.cfg* files from the AlexeyAB's Github (Bochkovskiy, 2019). Specifically, the parameter 'classes' was set to 3, and the corresponding parameter 'filter' was changed to the number $(classes + 5) \times 3$. We set the maximum iteration number as 2000 $\times$ classes, which was 6000 in our case. Different input sizes of resolution were tuned in our *yolov4-tiny.cfg* file. We also tuned the parameter 'random' from 1 to 0 whenever a memory issue was encountered during training. On the other hand, *train.txt* and *test.txt* were used to guide the custom model to learn the object images in our training dataset and validation dataset respectively in a specific order. Meanwhile, *obj.names* and *obj,data* files were to instruct the custom detector about each specific directory during training. Note that the 'backup' path under the *obj.data* file was the location to which we saved the weights files of our model throughout training. By having this backup location, we could prevent the loss of previous weights files in case a training process accidentally ceased. Importantly, we followed the suggestion of Feng et al. (2021) in performing 'training from scratch' method, which was proved to help minimize the false negative of training results.

For reference, throughout the training, our network parameter values are as follows: batch size= 64, momentum=0.9, decay=0.0005, learning rate=0.00261, threshold=0.5, and iteration number=6000.

Regarding the steps of compilation, for the sake of utilizing GPU on our local computer to enhance the training speed and quality due to its nature of large memory bandwidth, we first built OpenCV libraries with CUDA and CUDNN. Subsequently, Darknet was installed for training. Another channel was the training on 'Google Colab', which leverages Google's GPU to speed up the training process. Both methods achieve the same performance.

4.1.2 Bounding Box Prediction

With a custom model being trained, we then utilize the model for object detection. The model is expected to produce 2D bounding boxes on every single frame of the streaming FoV.

As introduced in Section 2.1.2, the k-means clustering, and anchor-box mechanism were adopted in predicting bounding box on objects since the change of YOLOv2. The method of using anchor box to predict bounding box could increase the average IOU for each grid cell, and thus enhance the overall accuracy of object localization (Redmon and Farhadi, 2017). Anchor box resolved the problem of YOLOv1, in which one grid cell could only detect one object but it was unable to detect the other adjacent object within same grid cell. Anchor-box mechanism could detect multiple objects in one grid cell.

By referring to the ideas of Gao et al. (2019), the details of anchor-box-based bounding box prediction of YOLOv4-Tiny are explained as follows:

1. There is a pre-defined number and shape of anchor boxes on each grid cell through k-means dimension clusters method. For example, if the default number is 3, the YOLOv4-Tiny outputs 6 x 6 x 3 anchor boxes on a 6 x 6 feature map. The center of the anchor box is always located at the center of its respective cell. The shape is normally rectangular size in different orientation and aspect ratio. The Fig 4.1.2(a) illustrates 3 anchor boxes (assumed 3 for simplicity of explanation here, it could be any number in reality) on each grid cell.

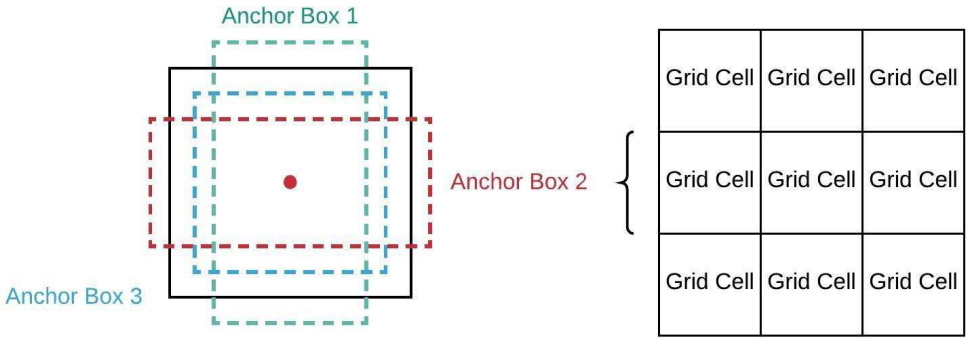


Fig. 4.1.2 (a) anchor boxes in each grid cell (Li, 2019)

2. Every anchor box predicts class and objectness. Among all numbers of anchor boxes on different grid cells, only the anchor boxes which are predicted to contain object (i.e. objectness = 1 with certain confidence score) would be keep. The Fig. 4.1.2(b) displays an example, 3 discrete color of anchor boxes 'aims' for different class of target object.

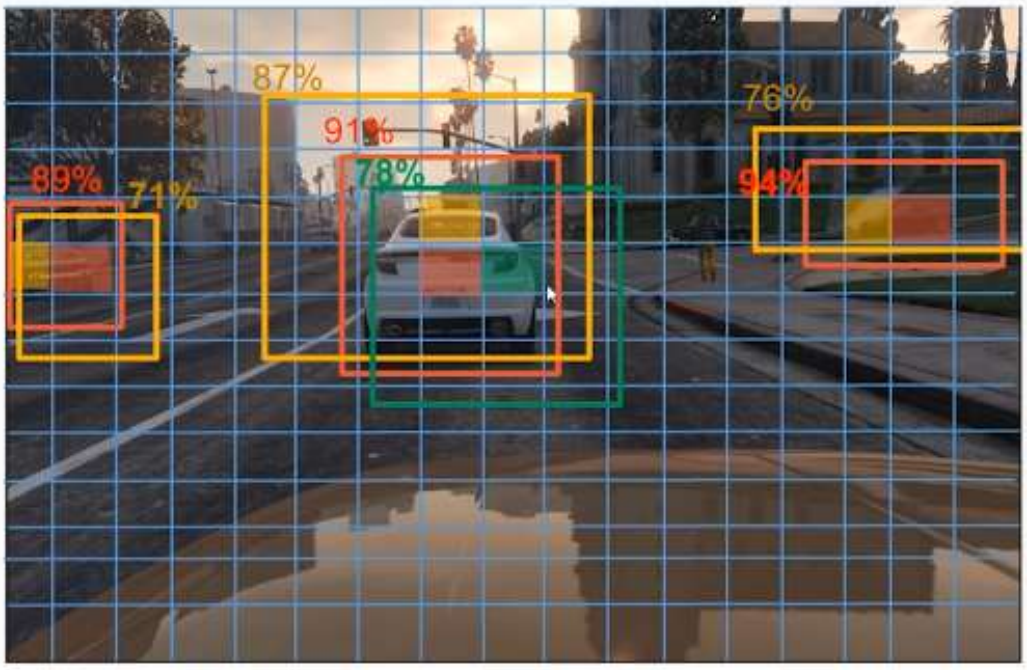


Fig. 4.1.2 (b) anchor boxes prediction (Ramo, 2019)

3. Among the remaining anchor boxes, only the anchor boxes that have the highest similarity and closest shape to the shape of ground-truth box of target object would be

keep as positive anchor boxes for further processing. In other words, the selection of the anchor box depends on the confidence score output of the network and the following non-max suppression (NMS) technique, or more explicitly, the highest IOU between the ground-truth box and the selected anchor box. (Fig. 4.1.2 (c))

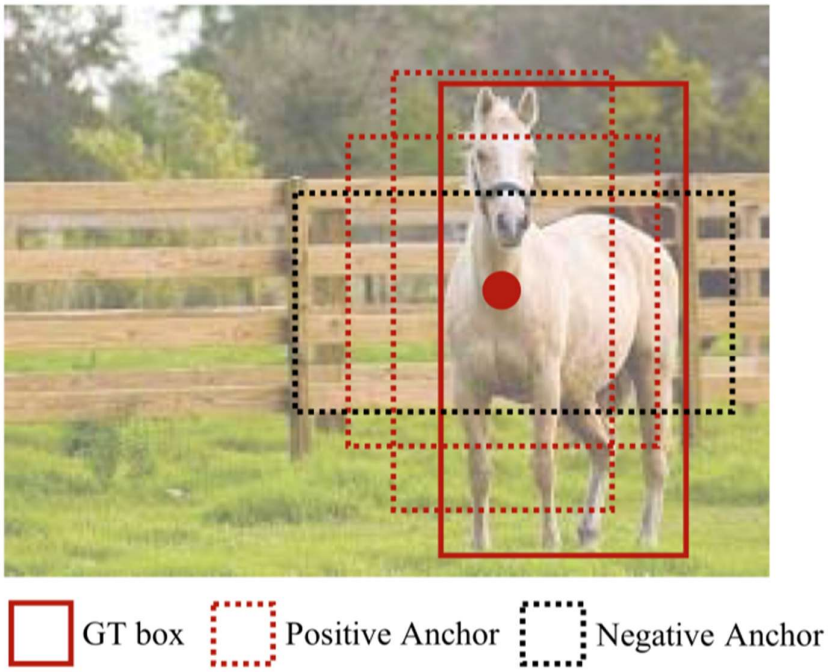


Fig. 4.1.2 (c) Selection of positive anchor boxes (Kong et al., 2020)

4. After acquired the anchor boxes for particular object, the anchor boxes with score values higher than the set confidence threshold values are further transformed to the final predicted bounding box by using a parameter regression function.
5. According to Redmon and Farhadi (2018), YOLO adopted the following computation in transforming the anchor box to the predicted bounding box. Zhong et al. (2020) pointed out that 'YOLO' framework uses absolute offset prediction. One anchor box generates one bounding box.

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w^{e^{tw}}$$

$$b_h = p_h^{e_{th}} \quad (4.1.2\ a)$$

- t_x , t_y , t_w , t_h are coordinates of predicted bounding box in terms of x position, y position, width, and height, which all have not transforming to finalized bounding box coordinates.
- c_x , c_y are the offset of cell from the top left corner of the image.
- p_w is the width of the predicted prior anchor box, p_h is the height of the predicted prior anchor box.
- σ is the sigmoid function applied to constrain the offset range between 0 and 1.
- b_x , b_y , b_w , b_h are the finalized parameters of bounding box, where b_x and b_y are the center coordinates, b_w and b_h are the width and height respectively (Fig. 4.1.2 (d))
- The Fig. 4.1.2 (e) demonstrates the relationship between anchor box and finalized bounding box.

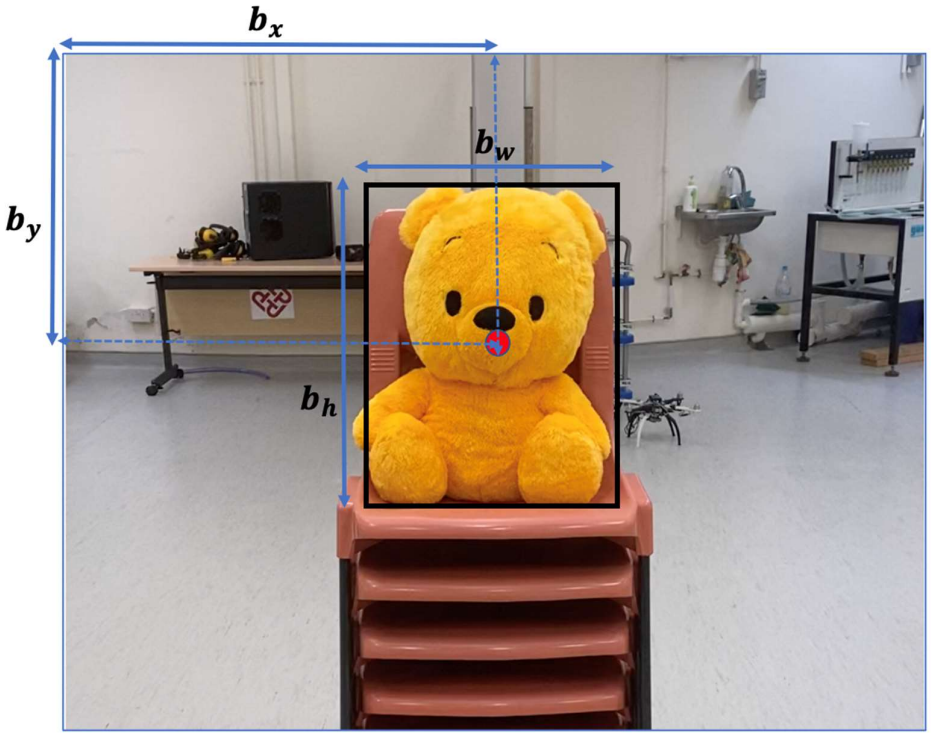


Fig. 4.1.2 (d) Bounding box coordinates in image plane

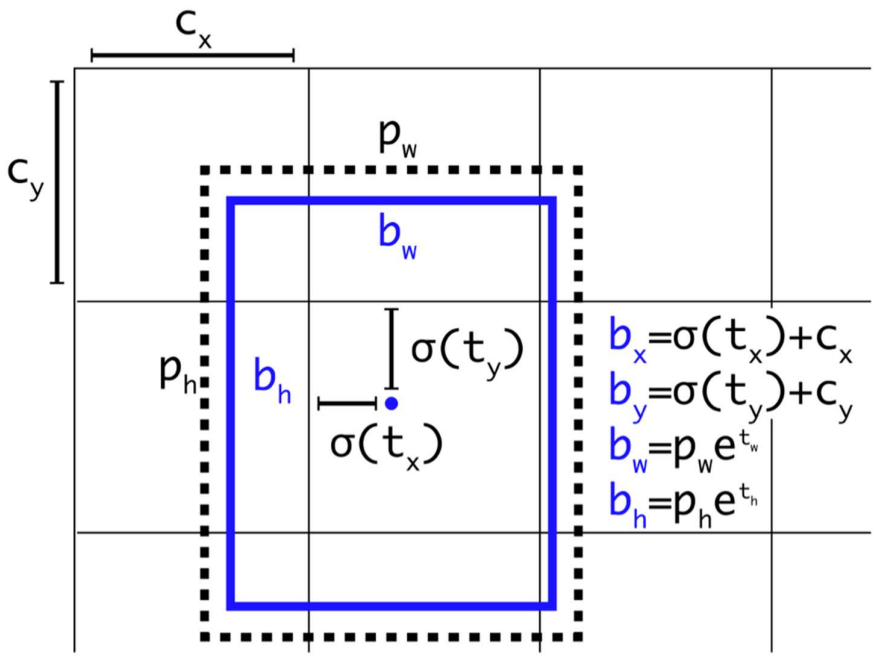


Fig. 4.1.2 (e) Transformation from anchor box to bounding box
(Redmon and Farhadi, 2018)

6. Among all bounding boxes, the bounding boxes with highest confidence score would be shown and kept as the predicted location of target object. The computation of prediction follows the same mechanism as explained in Section 2.1.2. The confidence threshold value was manually decided by our team, 0.25 was chosen due to the consideration of motion blur phenomenon on real-time UAV aerial images while employing medium mAP YOLOv4-Tiny detector.
7. Lastly, we saved the coordinates of the bounding box which contained the target object as box_{yolo} and further forward to pose estimation step.

4.2 Pose Estimation of Object

4.2.1 Frame Transformation

After acquiring bounding box information from above, we then consider an inner box box_{yolo} of the YOLO-generated bounding box box_{yolo} for depth measurement. The inner box is set to be:

$$box_{depth}.x = box_{yolo}.x$$
$$box_{depth}.y = box_{yolo}.y$$

$$box_{depth}.w = 0.2 \times box_{yolo}.w$$
$$box_{depth}.h = 0.2 \times box_{yolo}.h \quad (4.2.1\ a)$$

We set the scaling value at 0.2, which is lower than the previous work done by Feng et al. (2021). The reason for a smaller scaling value is due to the application of YOLOv4-Tiny, whose generated bounding boxes, when compared with YOLOv4, could be relatively unstable between frame and frame. As Feng et al. (2021) opted YOLOv4 as their object prediction method, we thus set a lower scaling value.

The acquired box_{yolo} will then play as the Region of Interest (ROI) for pose estimation. From the depth image acquired from the RGB-D camera, we first filtered out the unfilled pixels in ROI (whose value of depth is zero) with the filter function Find-Non-Zero in OpenCV (n.d.) and averaged the remaining depth data in box_{yolo}. We then assumed the averaged depth value as the distance between the observer and the target object. Subsequently, we conducted pose estimation by getting the global pose of the object through transforming the measured pose ($x_{2D}, y_{2D}\ and\ z_{dept}\ \in X_i^{2D}$) to world coordinate system. The frame transformation equations are as follows (Collins, 2007 & Feng et al., 2021):

$$\begin{bmatrix} X_i^W \\ 1 \end{bmatrix} = T_B^W T_C^B \begin{bmatrix} X_i^C \\ 1 \end{bmatrix}, T_B^W T_C^B \in SO(3) \quad (4.2.1\ b)$$

Where X_i^W is the object pose vector in world frame (x,y,z), T_B^W is the transformation matrix between world and body, T_C^B is the transformation matrix between body and camera, and X_i^C being the object pose vector in camera frame.

In particular, for the vector in camera frame, we denote $x_C = X_i^C(1), y_C = X_i^C(2)$ & $z_C = X_i^C(3)$:

$$x_C = X_i^C(1) = X_i^{2D}(3) \times \frac{X_i^{2D}(1) - T_x(baseline)}{f_x}$$
$$y_C = X_i^C(2) = X_i^{2D}(3) \times \frac{X_i^{2D}(2) - T_y(baseline)}{f_y}$$
$$z_c = X_i^C(3) = X_i^{2D}(3) = average\ depth\ of\ the\ box_{depth} \quad (4.2.1\ c)$$

while the transformation matrices being:

$$T_C^B = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_B^W = \begin{bmatrix} r_{11} & r_{12} & r_{13} & o_x \\ r_{21} & r_{22} & r_{23} & o_y \\ r_{31} & r_{32} & r_{33} & o_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.1.2\ \text{d})$$

in which r_{ij} is the element in the rotation matrix of the attitude of the observer, and o_x, o_y & o_z are the current position of observer (UAV) with respect to the world frame. The rotation of a coordinate frame is usually expressed in either rotation matrix or quaternion representation.

For further detailed information of the coordinate transformation, we refer readers to Collin's presentation (2007). Below (Fig. 4.2 (a) & Fig. 4.2 (b)) shows the coordinate transformation between camera, body and world frame. For the rest of the report, the coordinate transformation implementation would be referring to the presented content here in Section 4.2.

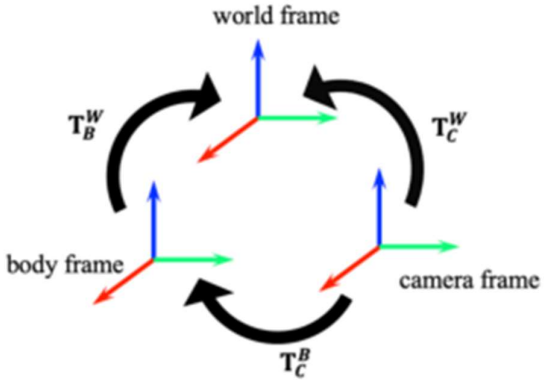


Fig. 4.2 (a) (Feng et al., 2021)

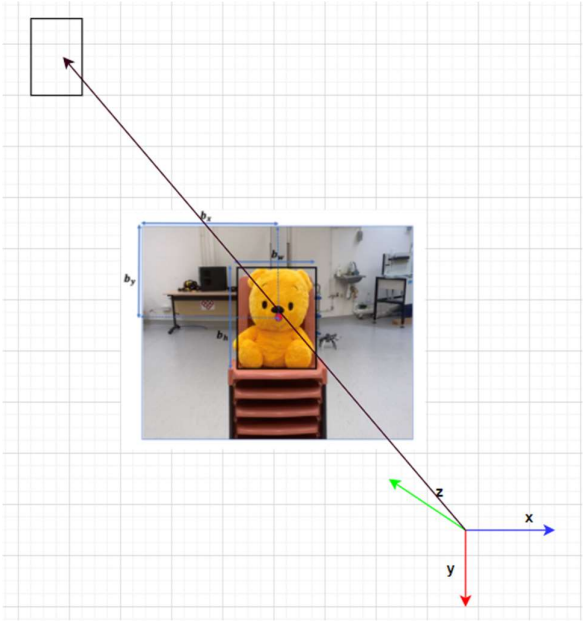


Fig. 4.2 (b)

4.2.2 IIR filter

In addition to above, due to the high-frequency noise as well as the measurement error of the RGB-D camera, we utilized Infinite Impulse Response (IIR) filter in order to have a more accurate object pose estimation (Grout, 2011 & Feng et al., 2021). The IIR filter is basically a filter that recursively computes the output (in our case, the object's pose estimation) with previous derived input and output (Grout, 2011), so that the estimation could converge gradually. To deal with a noisy sensor, such method could give us an estimation that will not deviate extensively from the ground truth, as our pose estimation is not only based on the current one measurement. The IIR equation could be written as:

$$X_{i_{avg}}^{W} = \alpha X_{i}^{W} + (1-\alpha) X_{i-1_{avg}}^{W} \qquad (4.2.2\ a)$$

In which, $X_{i_{avg}}^{W}$ is the estimated pose at time step i, while $X_{i_{i-1av}}^{W}$ being the estimated pose at time step i-1. X_{i}^{W} is the current measured pose and α is the IIR filter parameter.

Based on the work done by Feng et al. (2021), we assumed the IIR filter parameter to be 0.9. We found that the suggested parameter value could give us a converging result on object observation and a small error with the ground truth. Further experiment results are presented in Section 6.

4.3 Obstacle Avoidance with Edge-Detection Planner

The main idea of designing an obstacle avoidance local planner is to allow a robot to safely traverse through a 3-D space without collision. As mentioned above in the literature review Section 2.3, most of the UAV path planning methods could be classified into two kinds: the global path planning which utilized pre-known environmental information and the local path planning that respond to local obstacle information retrieved from sensors. Therefore, after our investigation on several previous works, we came up with a local obstacle avoidance algorithm that aims to robustly avoid column-wise obstacles; in which it utilizes image processing techniques and depth data from the RGB-D camera. The overall algorithm is presented below in **Algorithm 1**, where we predominantly try to search for a new 3D waypoint on the observed image frame when encountering an obstacle and further generate motion primitives for the quadrotor. Our algorithm is inspired by Bug algorithms, which will try to return to an imaginary m-line after the avoidance of an obstacle (Lumelsky & Stepanov, 1987). In addition to that, when no obstacle is observed, the robot will try to reach the global waypoint P_{goal} (destination) via the shortest route (m-line) calculated. Section 4.3.1 to 4.3.3 will discuss the front-end work of our Edge Detection Planner, while Section 4.3.4 will focus on the back-end planning.

Algorithm 1 Edge Detection Planner

```
1.  while true do
2.    if on m-line do
3.      collision check
4.    end if
5.    if obstacle found then
6.      trigger way point search and avoid
7.    else
8.      go to P_goal with v_max
9.    end if
10.   forward motion primitives to flight controller of quadrotor
11. end while
```

4.3.1 Collision Check

To assess whether the UAV would be crashing into an unknown obstacle, a 3-D future motion point (P_{f_i}) (where i denotes the state at different time step) will be derived; based on the UAV's dimension, an according prediction box ($box_{p_{i(W)}}$) will also be generated. The future motion point (P_{f_i}) and prediction box ($box_{p_{i(W)}}$) are the predicted UAV state at future time step, which is denoted as discrete time step index t_{i+k}, and additionally, the prediction box ($box_{p_{i(W)}}$) is set to be having a dimension of an enlarged UAV for keeping the UAV to maintain a safety distance from the obstacles. Both P_f and $box_{p_{i(W)}}$ will then be transformed from world coordinate system (X_i^W) to camera coordinate (X_i^C) through transformation matrices T_B^W and T_C^B, and both will be further projected onto image plane. The frame is derived iteratively from the aerial image during the manoeuvrer. An acquired camera frame box_{p_i} on image plane is shown as below in Fig. 4.3.1 (a).

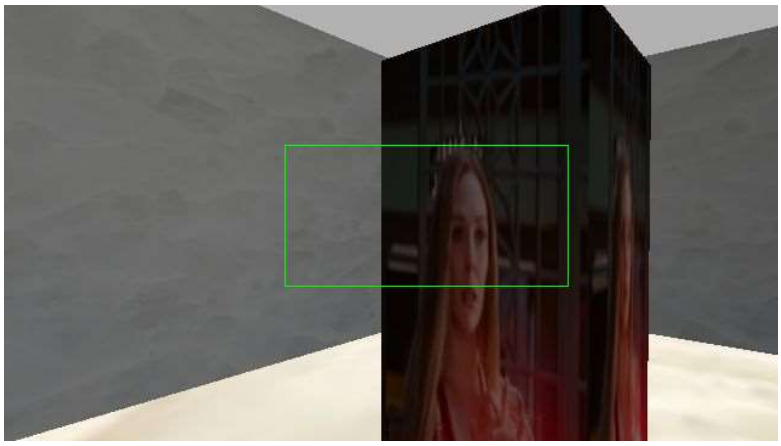


Fig. 4.3.1 (a) the green rectangle is a prediction of the UAV state that projected on to the FoV's image plane

With Box_{P_i} being projected onto image plane, we then consider box_{p_i} as ROI and access the depth data and justify whether the depth value of any pixel in the ROI is smaller than P_{f_i}'s value on the optic axis ($X_i^C(3)$). $X_i^C(3)$ could be understood as the expected minimum depth. Particularly, due to the zero-valued hole regions, we also applied Find-Non-Zero filter (OpenCV, n.d.) to filter out the unfilled depth and do collision check based on the remaining data. Below (Fig. 4.3.1 (b)-1 & 2) visualizes the collision check with Box_{p_i}.

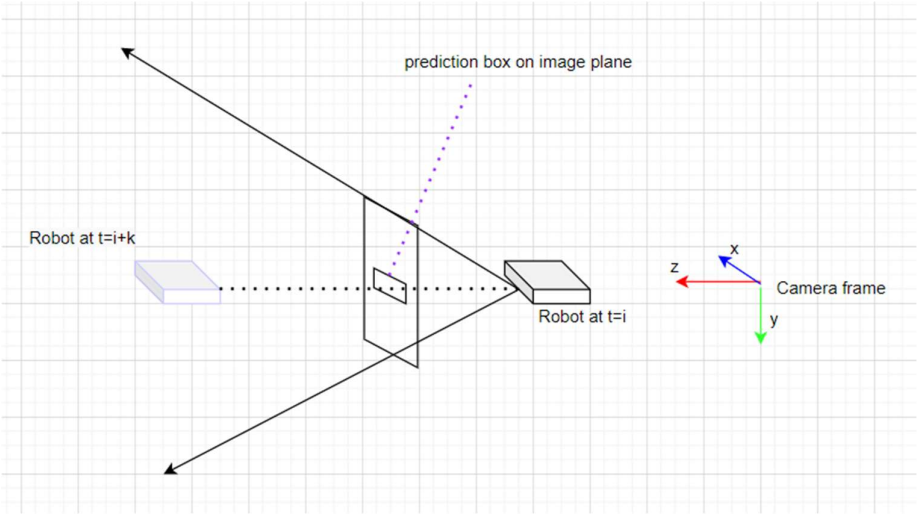


Fig. 4.3.1 (b)-1 the figure shows when no obstacle is detected

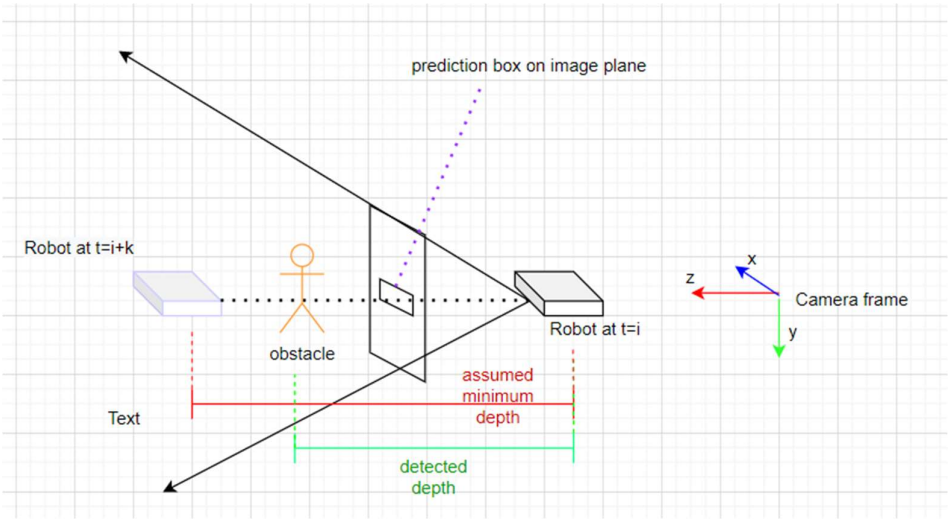


Fig. 4.3.1 (b)-2 an obstacle is detected when an obstacle stands between the predicted object pose and current pose. The Edge Detection Planner will be triggered.

If the system assumed an encounter of obstacle, the local Edge Detection Planner would be triggered. Below shows the algorithm of collision check.

Algorithm 2 collision check

Remark: D as depth data from RGB-D sensor, P_{f_i} as future motion point, W as world frame, C as camera frame, Box_{f_i} as the prediction box around P_{f_i} (based on UAV dimension)

1. **Input:** D
2. **while** *true*: **do**
3. get P_{f_i} at $t = i + k$ (i being the current time, while $i + k$ being the future time step)
4. get d_{min}, the expected minimum depth of P_{f_i}
5. Get $box_{p_i(W)}$ based on UAV dimension and P_{f_i}
6. transform $P_{f_i}(W)$ to $P_{f_i}(C)$
7. transform $box_{p_i(W)}$ to $Box_{f_i}(C)$
8. set D', the depth data of ROI $Box_{f_i}(C)$
9. **for** d_i in D'
10. **if** $d_i < d_{min}$
11. **return** obstacle found as *true*
12. **end while**
13. **end if**
14. **end for**
15. **end while**

4.3.2 Waypoint Search

With an obstacle being detected, the local planner will be triggered. Based on the afore acquired box_{P_i}, we try to locate a reference point P_{ref} for waypoint P_{wp} generation. Below illustrates the overall concept of our local planner.

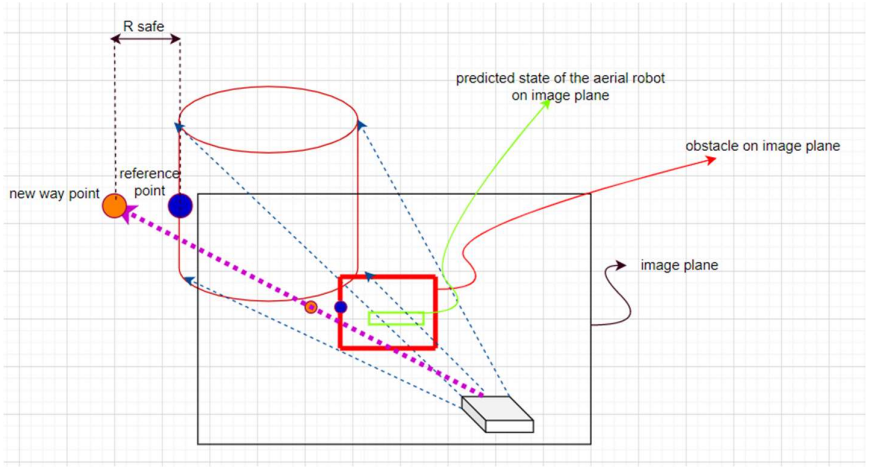


Fig. 4.3.2 (a) Our proposed local Edge Detection Planner: the basic idea is to find a reference point with image process technique (edge detection) on the frame received from the RGB-D sensor. The reference point (blue dot) is then being used for waypoint (orange dot) search.

To get P_{ref}, a Canny Edge Detector is applied. The Canny Edge Detector is predominantly an image processing technique to extract the structural information from a certain image, through which the image's intensity gradient is calculated for the detection. For our implementation, we utilized the OpenCV function library, where the gradient equation of every pixel is considered to be (OpenCV, n.d.):

$$Edge\ Gradient\ (G) = \sqrt{G_x^2 + G_y^2} \tag{4.3.2 a}$$

$$Angle(\theta) = \tan^{-1}\left(\frac{G_x}{G_y}\right) \tag{4.3.2 b}$$

After acquiring the gradient value and the corresponding direction, the OpenCV function will then conduct Non-maximum Suppression and Hysteresis Thresholding base on the entered maximum value and minimum value. Non-maximum Suppression basically checks whether the pixel possess the maximum value among its neighbour on the gradient direction, i.e., the derived θ, and suppress other non-maximum pixels to zero. As for hysteresis thresholding, the function will base on the predefined maximum value ($maxVal$) and minimum value ($minVal$) and justify whether the pixel should be kept or eliminated. For pixels with a larger gradient value than maximum value, they will be considered as 'edge detected', while those with a smaller gradient value than the minimum value will be disposed. Additionally, for the pixels with values between $maxVal$ and $minVal$, they will be classified in terms of their connectivity with the 'edge detected': if it is connected with those above $maxVal$, it will be deemed as 'edge'; otherwise, it will be discarded as 'non-edge'. The final result from the detector will be displayed in a binary image, with pixels denoted as 'edge' being white. Below (Fig. 4.3.2 (a) & Fig. 4.3.2 (b)) shows an instance of an image being processed by Canny Edge Detector.

Fig. 4.3.2 (b) the original image

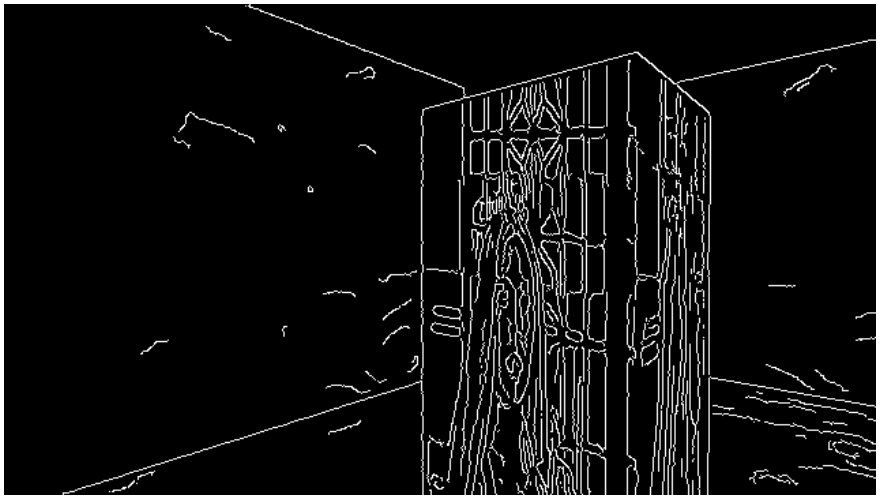


Fig. 4.3.2 (c) an image after Canny Edge Detection. In this case, we set maxVal as 20, while minVal as 10, which are both relatively low, since we expect a more sensitive detector so that no crucial edge features would be obliviated.

For more information on the algorithm, we refer readers to OpenCV documentation (n.d.) and Canny's (1986) pioneering work.

After retrieving the result from edge detection, we retain the pixels which are selected as "edge" and whose depth value is $d < d_{predefined}$, while eliminating the others. We also clear the non-filled pixels with Find-Non-Zero function (OpenCV, n.d.). The value $d_{predefined}$ is set differently according to different environments and it is usually empirically determined.

Furthermore, for simplicity, we only consider pixels, whose x coordinate and y coordinate are respectively near the x centre and y centre of Box_{P_i}, to be our potential reference points, which gives us a cross-like region (as depicted in Fig. 4.3.2 (e)). The selection process is considered to be:

$$\begin{gathered} select\ pixels \\ whose\ \left|pixel.x - box_{P_i}.x\right| < 0.1 \times box_{P_i}.w \\ or \\ whose\ \left|pixel.y - box_{P_i}.y\right| < 0.1 \times box_{P_i}.h \end{gathered} \tag{4.3.2 c}$$

To further screen out the unfeasible reference points, we checked the pixels surroundings to justify whether they could act as a reference point for waypoint generation:

$$\begin{gathered} select\ pixels\ (P_c) \\ (neighbour\ pixel\ (in\ both\ x\ and\ y\ direction)\ denoted\ as\ P_c') \\ where \\ P_c'.depth - P_c.depth > 0.1\ m \end{gathered} \tag{4.3.2 d}$$

After filtration, we then calculated the distance between the candidate pixels and the centre of box_{P_i}, and deemed the pixel with the shortest distance as the final reference point P_{ref}. **Algorithm 3** shows the process of reference point search:

Algorithm 3 reference point search

Remark: PX as pixels data from RGB-D sensor, Box_{f_i} as the prediction box around,

1. **Input:** PX, Box_{f_i}
2. apply Canny Edge Detection on PX, and hence get PX_{edge}
3. **for** px_i in PX_{edge}
4. Remove unfilled px_i ($px_i.depth = 0$)
5. **end for**
6. **for** px_i in PX_{edge}
7. keep px_i within the cross region
8. **end for**

9. **for** px_i in PX_{edge}
10. remove unfeasible px_i
11. **end for**
12. set $px_c = Box_{f_i}.centre$
13. **for** px_i in PX_{edge}
14. **do** $d = \|\overline{px_i\ px_c}\|$
15. **if** $d_{min} = \|\overline{px_i\ px_c}\|$
16. set $px_{return} = px_i$
17. **end for**
18. **end if**
19. **end for**
20. **return** px_{return}

The following images (Fig. 4.3.2 (d) to Fig. 4.3.2 (h)) further display the first stage of our waypoint search method, i.e., reference point search. The images are retrieved from real-time simulation in Gazebo.

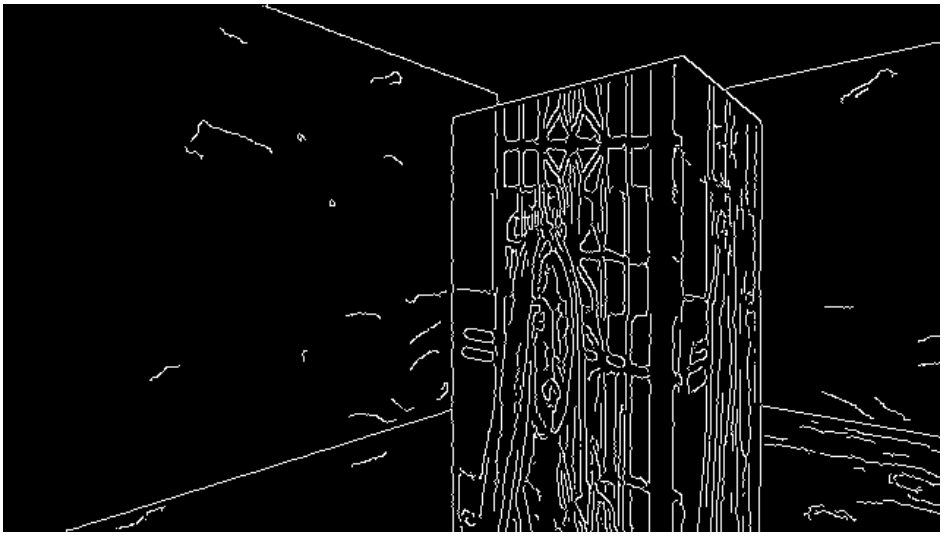


Fig. 4.3.2 (d) the original Canny Edge detected image.

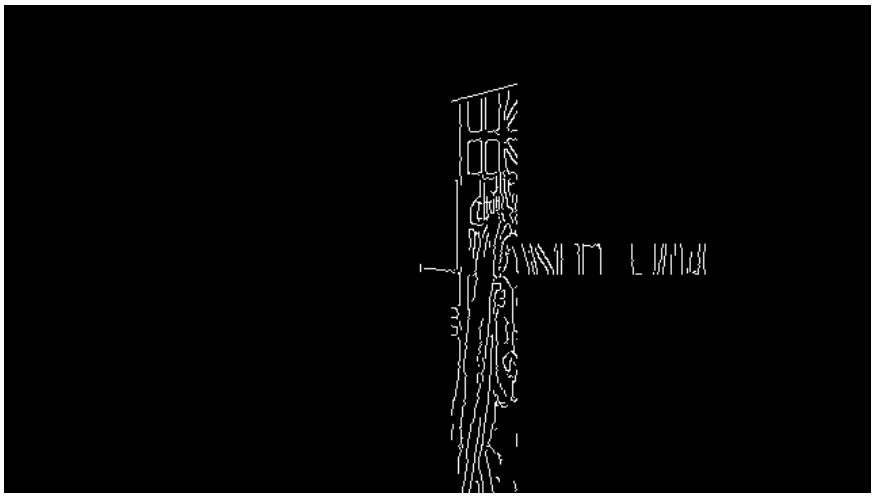


Fig. 4.3.2 (e) the "cross-like region" result after removing pixels whose depth ($d > d_{predefined}$) as well as selecting only the pixels that locate near the x centre or y centre of Box_{P_i}.

In addition to above, we then attempted to get the centre of gravity of the obstacle. We did it by selecting pixels with depth $d < d_{predefined}$, further projected them onto a blank image, and compared the "centre of gravity" of pixels of the newly derived image with the centre of box_{P_i}. This could then suggest the direction of the movement for the robot (i.e., whether the UAV should pass the obstacle from up, down, left or right direction).

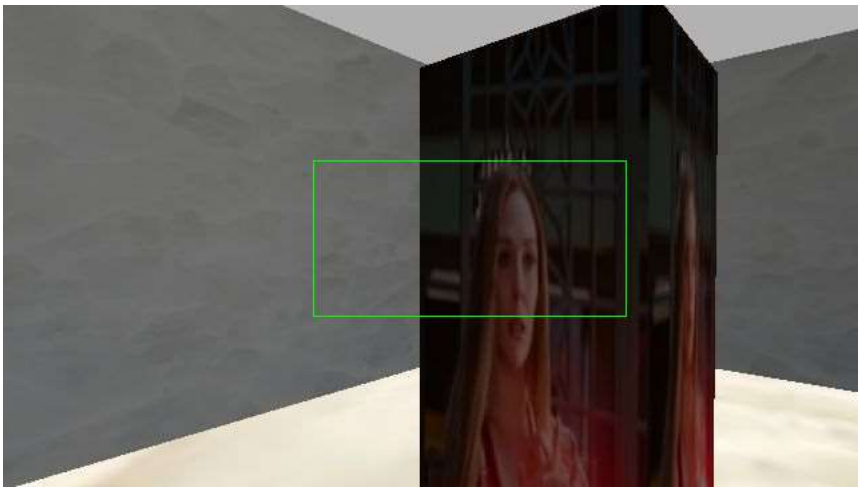


Fig. 4.3.2 (f) the original observed environment

Fig. 4.3.2 (g) the pixels whose depth $d < d_{predefined}$

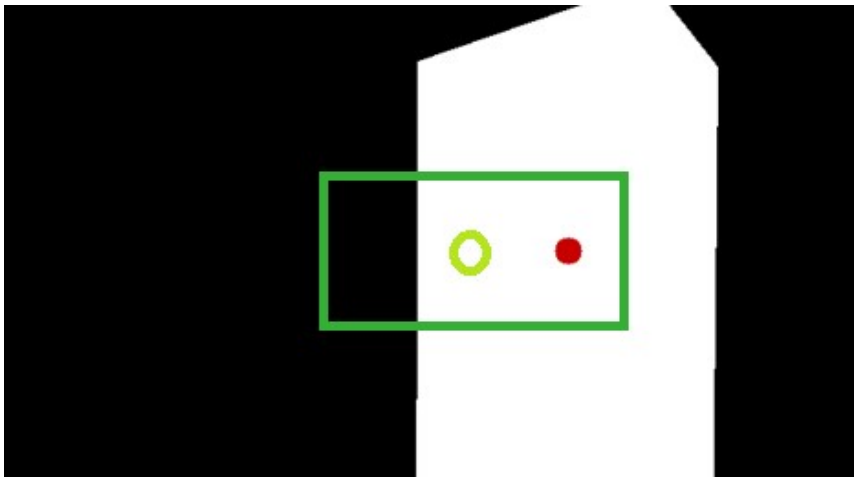


Fig. 4.3.2 (h) red dot being the C.G. of the obstacle. the derived CG will then be used for determination of moving direction. In this case, as the centre of the predicted state box_{P_i} is relatively left to the CG, hence, it is most likely to generate a waypoint that allows the UAV to move from the left.

As mentioned, the result P_{ref} will act as the reference point for waypoint search. P_{ref} will first be converted into X_i^B in body coordinate; P_{wp} will then be secured by adding or deducting R_{safe} on either the y axis or z axis of P_{ref} in the body coordinate, depending on the suggested

direction from previous step. P_{wp} (X_i^B) will then be transformed to $P_{wp}(X_i^W)$ by T_B^W , and we hence get the new waypoint.

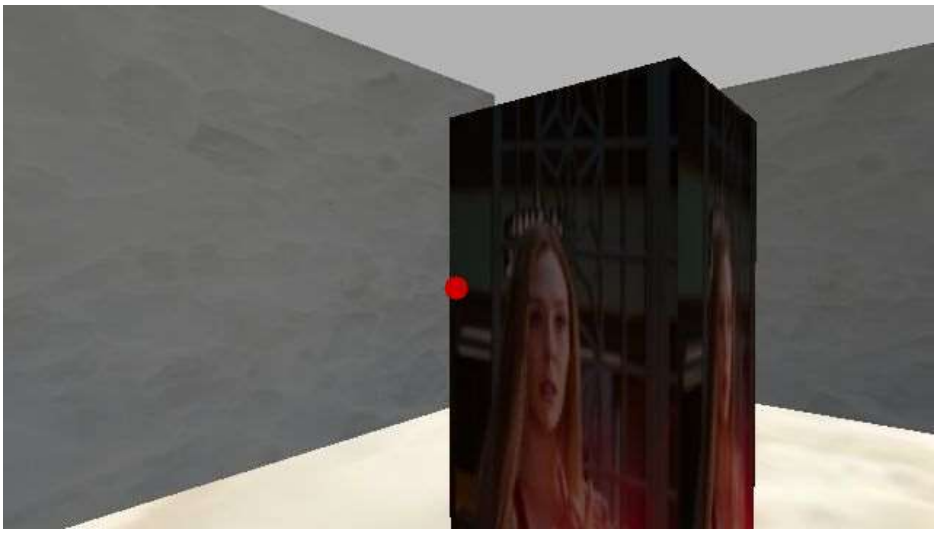


Fig. 4.3.2 (i) the final reference point, the red dot.

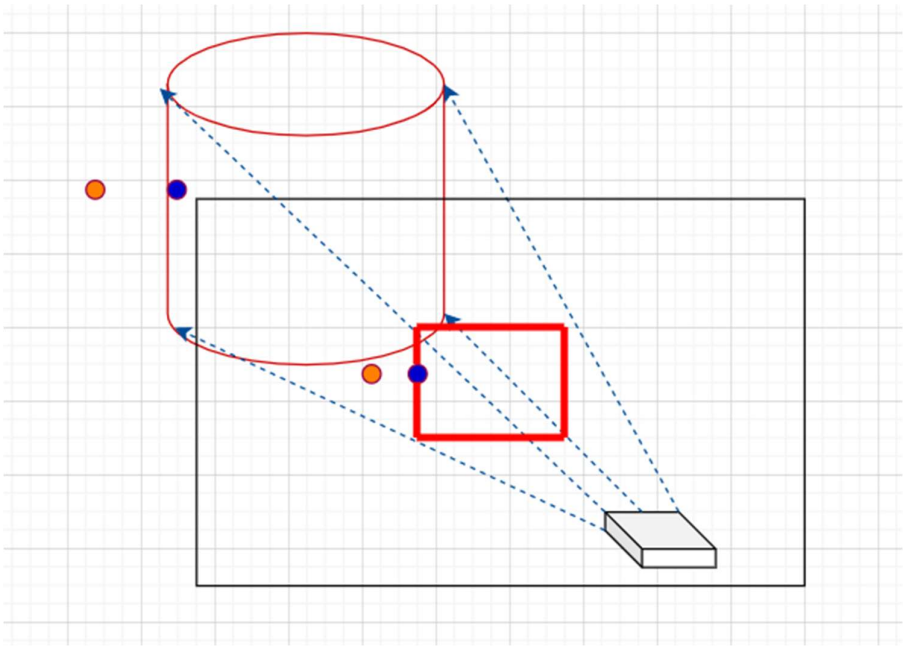


Fig. 4.3.2 (j) with the final reference point, we then get a new waypoint. The blue points show the found reference point, while the orange point displays the newly generated waypoint. The distance between the 2 points is set to be R_{safe}

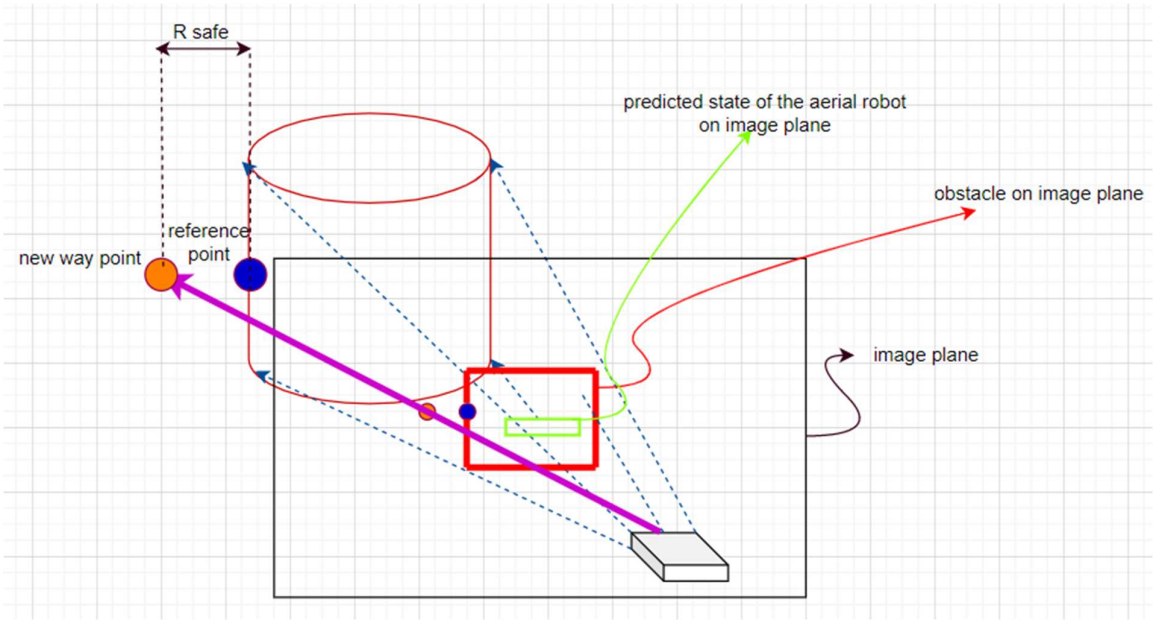


Fig.4.3.2 (k) the overview of local Edge Detection Planner

4.3.3 Safety Guarantee

To guarantee a collision free motion around the obstacle, we assume every met obstacle column-wise in a conservative manner. Therefore, with the P_{wp} being obtained from waypoint search, we then calculated a P'_{wp} by assuming P'_{wp} being $W_{obstacle}$ away from P_{wp} on the x axis in the body coordinate system. $W_{obstacle}$ is the width of the obstacle, which is calculated based on the observed image. With the additional P'_{wp}, we then set $P_{current}, P_{wp}, P'_{wp}$ and P_{aft} as the selected avoiding waypoints, in which P_{aft} is the waypoint located after the obstacle. P_{aft} is assumed to be on the m-line of $P_{current}$ and P_{goal}, and is $\sqrt{2}W_{obstacle}$ away from P_{wp} on the x axis in the body coordinate system. Fig. 4.3.3 below shows the safety guarantee waypoint setting of the Edge Detection Planner.

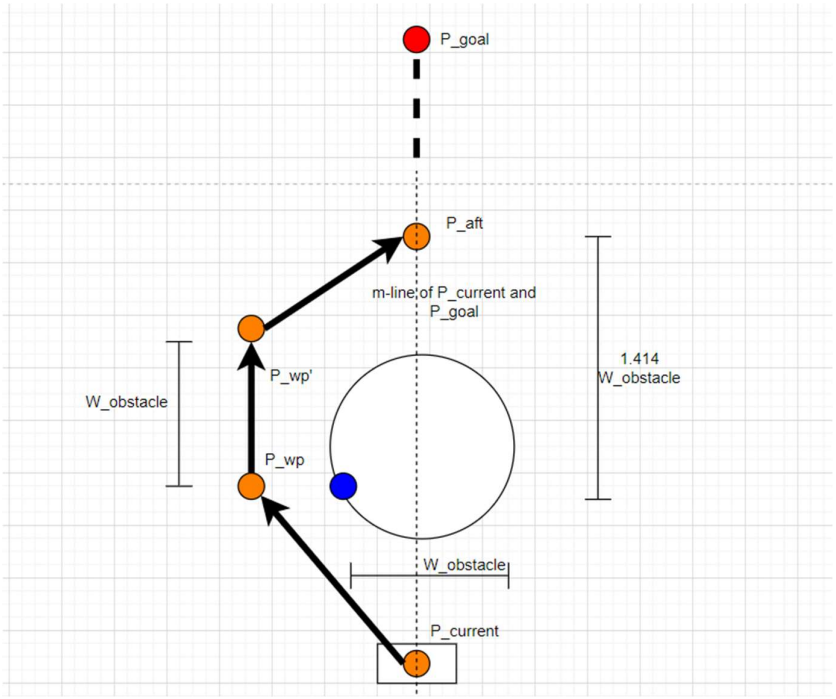


Fig. 4.3.3 the proposed safety guarantee method of ours.

Our safety guarantee method could be described as **Algorithm 4** below:

Algorithm 4 safety guarantee

Remark: W as world frame, B as body frame

1. **Input:** $P_{current}, P_{wp}$ & $W_{obstacle}$
2. get $P'_{wp}(B)(1) = P_{wp}(B)(1) + W_{obstacle}$
3. get $P'_{wp}(B)(2) = P_{wp}(B)(2)$
4. get $P'_{wp}(B)(3) = P_{wp}(B)(3)$
5. transform $P'_{wp}(B)$ to $P'_{wp}(W)$
6. get $P'_{aft}(B)(1) = P_{wp}(B)(1) + \sqrt{2}W_{obstacle}$
7. get $P'_{aft}(B)(2) = P_{current}(B)(2)$
8. get $P'_{aft}(B)(3) = P_{current}(B)(3)$
9. transform $P'_{aft}(B)$ to $P'_{aft}(W)$
10. $P \leftarrow P_{current}, P_{wp}, P'_{wp}$ & P_{aft}
11. **return** P

4.3.4 Optimized Trajectory

To consider the robot's kinetic feasibility in aggressive flights, instead of commanding UAV to move in straight lines when avoiding obstacles, with the acquired waypoints, we applied optimization method for trajectory generation. With the trajectory being optimized as piece-wise polynomials, it is believed that the aerial robot could travel with a smooth curve and with less control effort. Based on works of Mellinger and Kumar (2011) as well as Richter et al.'s (2016), the below presents the optimization of our planner.

4.3.4-1 Quadratic Programming

A Quadratic Programming (QP) problem usually involves multiple variables and aims to minimize/maximize a quadratic objective function such that the variables satisfy some linear equality or inequality constraints. Eventually, a n-dimensional vector x, which represents a set of values of all decision variables that minimize the cost function (i.e., objective function), will be output. The minimum snap or minimum jerk problems, as mentioned in Section 2.3, will be formulated into a multivariate QP problem, which can be generalized into the following form:

$$Minimize\ f(x) = \frac{1}{2} * x^T * H * x + q^T * x$$
$$s.t. Ax = a\ Bx \leq b \qquad (4.3.4\ a)$$

where f(x) is the objective function; x is a vector containing all decision variables, H is a matrix that contains the coefficients of all quadratic terms as well as bilinear terms in the objective function, and q is a vector that contains the coefficients of all first-order terms in the objective function. As for $Ax = a$ & $Bx \leq b$, $Ax = a$ incorporates all of the linear equality constraints while $Bx \leq b$ incorporates all of the linear inequality constraints.

Regarding a low-dimensional QP problem, the objective function can be readily solved mathematically, while for a high-dimensional problem involving numerous variables, a QP solver such as MATLAB *quadprog()* function or C++ OOQP package (Gertz & Wright, 2003) can be utilized to acquire the solutions.

4.3.4-2 Minimum Jerk Trajectory Generation

Instead of applying trajectory regeneration with minimum snap, we opted for minimum jerk (third derivation of position) as it could also produce a visualizable smooth trajectory for the

aerial robot and is suitable for visual tracking. The optimization problem could then be written in such piecewise form:

$$p(t) = \begin{cases} [1, t, t_2, t_3, \dots, t_n] * p_1 \ (t_0 < t < t_1) \\ [1, t, t_2, t_3, \dots, t_n] * p_2 \ (t_1 < t < t_2) \\ [1, t, t_2, t_3, \dots, t_n] * p_3 \ (t_2 < t < t_3) \\ [1, t, t_2, t_3, \dots, t_n] * p_4 \ (t_3 < t < t_4) \\ \dots \\ [1, t, t_2, t_3, \dots, t_n] * p_k \ (t_{k-1} < t < t_k) \end{cases} \quad (4.3.4\ \text{b})$$

$$where\ t\ here\ is\ on\ relative\ timeline$$
$$and\ P_i\ being\ the\ polynomial\ vector\ of\ every\ segment.$$

The cost function (objective function) would be thus defined as:

$$min \int_0^T (P^{(3)}(t))^2 dt \quad (4.3.4\ c)$$

minimum jerk algorithm is essentially trying to minimize the square of third derivative (i.e., the differentiation will be taken for six times), which necessitates the required minimum polynomial degrees for a single segment to be five (i.e., the highest order of the polynomial function shall be at least five).

For such an optimization problem, our main objective is to get the optimized polynomial vector. It could be done through solving a QP, by either using QP solver in MATLAB or QP solver libraries. Furthermore, for numerical stability and efficiency, based on the work of Richter et al. (2016), we adopted unconstrained closed-form solution to minimum jerk, where it could be defined as:

$$\min J = P^T Q P\ , where\ we\ wish\ to\ get\ P$$
$$while\ M_j P_j = d_j,$$

$$M\ being\ the\ mapping\ matrix\ between\ polynomicals\ and\ derivatives$$
$$hence, from\ above, P = M^{-1} d.$$
$$and, d = C^T \begin{bmatrix} dF \\ dP \end{bmatrix},$$
$$where\ dF\ is\ the\ contrained\ variable\ mattrix$$

$$and\ dP\ is\ the\ free\ variable\ matrix\ (unconstrained)$$
$$thus, dP\ here\ is\ unknown,$$
$$and\ we\ would\ like\ to\ solve\ it\ so\ that\ we\ could\ get\ d_j$$
$$and\ hence\ further\ get\ P_j$$

Hence, we write:

$$P = M^{-1}C^T \begin{bmatrix} dF \\ dP \end{bmatrix}$$
$$P^T = \begin{bmatrix} dF \\ dP \end{bmatrix}^T CM^{-T}$$

The whole equation will thus be:

$$\min J = \begin{bmatrix} dF \\ dP \end{bmatrix}^T CM^{-T}\ Q\ M^{-1}C^T \begin{bmatrix} dF \\ dP \end{bmatrix}$$

$$additionally, set\ CM^{-T}\ Q\ M^{-1}C^T = R$$

And thus,

$$\min J = \begin{bmatrix} dF \\ dP \end{bmatrix}^T CM^{-T}\ Q\ M^{-1}C^T \begin{bmatrix} dF \\ dP \end{bmatrix} = \min J = \begin{bmatrix} dF \\ dP \end{bmatrix}^T \begin{bmatrix} R_{FF}\ R_{FP} \\ R_{PF}\ R_{PP} \end{bmatrix} \begin{bmatrix} dF \\ dP \end{bmatrix}$$

Thus,

$$\min J = dF^T R_{FF} dF + 2dF^T R_{FP} dP + dP^T R_{PP} dP$$

We then get minimum by finding $\frac{\partial J}{\partial P} = 0$,

$$\frac{\partial J}{\partial P} = 0 + 2dF^T R_{FP} + 2R_{PP} dP = 0$$

And hence we solve:

$$dP = -R_{PP}^{-1} R_{FP}^T dF$$

$$we\ could\ get\ d = \begin{bmatrix} dF \\ dP \end{bmatrix}, and\ get\ P = M^{-1}d$$

After solving the closed form matrices equation, we then got the polynomial vectors for different segments for the motion primitives and further obtained the trajectories. The aerial robot would then follow the generated trajectory for object avoidance. In addition, the time allocation, which can be literally understood as "how much time should be distributed to each segment", is also considered a crucial step for trajectory generation. By taking reference of Shomin and Hollis' (2014) work, we "naively" applied the trapezoidal velocity time profile (Fig. 4.3.4 (a)), one of the standard profiles, for each segment, where the total time for single segment is calculated as:

$$notation:$$
$$v_m\ as\ the\ maximum\ velocity,$$
$$a\ as\ the\ constant\ acceleration,$$
$$d_s\ as\ the\ euclidian\ distance\ of\ the\ segment$$

$$t_1 = \frac{|v_m - v_0|}{a}$$
$$d_1 = (v_0 + v_m)\frac{t_1}{2}$$
$$t_2 = \frac{|v_m - v_f|}{a}$$
$$d_2 = (v_f + v_m)\frac{t_2}{2}$$
$$if\ d_1 + d_2 < d_s, t_m = \frac{d_s - (d_1 + d_2)}{v_m}$$
$$t_k = t_1 + t_2 + t_m$$
$$else\ t_k = t_1 + t_2$$

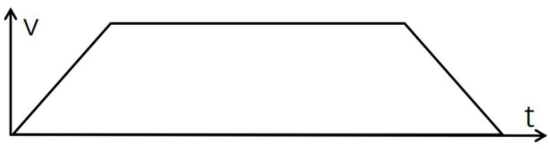


Fig. 4.3.4 (a) trapezoidal velocity profile

Fig. 4.3.4 (b) below further shows a demonstration of our minimum jerk trajectory generation.

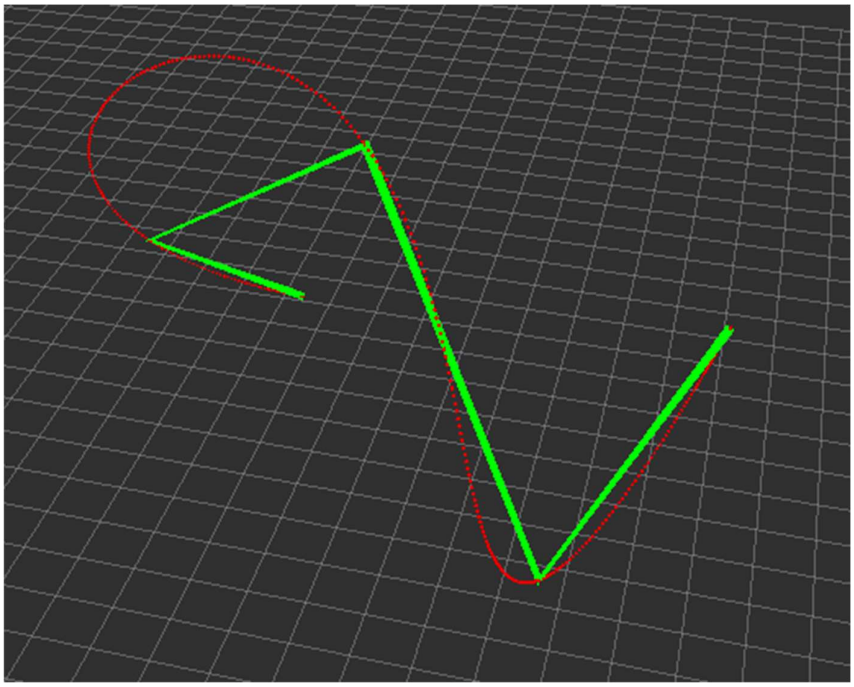


Fig. 4.3.4 (b)

4.4 Finite State Machine (FSM)

To perform object searching autonomously, we need to combine the 2 modules, i.e., the object detection and its pose estimation as well as the local planner, together. To merge them all in one, we adopt the Finite State Machine to describe our system.

The Finite State machine (FSM) is a computational model mainly for describing automatic machines states using a finite number of states and the respective transitions triggered by certain events. Gladyshev & Patel (2004) indicated that an FSM can assist to explore the state-space of a system and hence to determine all possible incidents that could happen in this system. Specifically in our case, FSM guides the actions of our UAV during the object searching module. In object searching (OS), our FSM is defined by six states: take off, sway and search, travel, avoid, locate and record, return. Fig. 4.4 depicts FSM with the states and the

corresponding transitions triggered by different inputs for the object searching module. The follow paragraphs explain each of the aforementioned states in more detail.

Take off: After calibrating the VICON and controller, as well as compiling all the program for UAV onboard computer, our UAV enters off-board control mode and takes off from the ground to achieve a hovering status. Once the hovering status is reached, the FSM is switched to the *Sway and Search* state.

Sway and Search: The UAV maintains its altitude and performs yaw motion in 360 degrees. At the same time, the onboard stereo camera will continuously search the target object. When the object is found, the FSM will transit into *Locate and Record* state. Otherwise, the UAV will see whether the current waypoint is the last waypoint or not. If it is the last waypoint on the pre-set path, the FSM will be switched to *Return* mode. On the other hand, if there is still waypoint to reach, the FSM will be switched to *Travel* state.

Travel: In this mode, the UAV will travel to the next pre-set waypoint. If there is an unknown obstacle on the way of the flight path, the FSM will transit into *Avoid* state. If no obstacle occurring, the FSM will be continue staying in *Travel* mode and switch to *Sway and Search* mode after arriving the waypoint.

Avoid: The UAV will avoid any unexpected obstacles using the path planning and trajectory generation algorithms mentioned in Section 4.3. Once the UAV has avoided all obstacles on its route, the FSM will switch back to *Travel* state, which will guide UAV to reach the next pre-set waypoint.

Locate and Record: The location information of the detected target object will be recorded in the onboard computer of the UAV in this mode. This state operates when the UAV is in hovering status. Subsequently, after a pre-set period, the UAV will check whether the current waypoint is the final waypoint, in order to determine whether the FSM will transit into *Return* state or *Travel* state.

Return: The UAV will travel back to the home position and land slowly at a constant vertical speed.

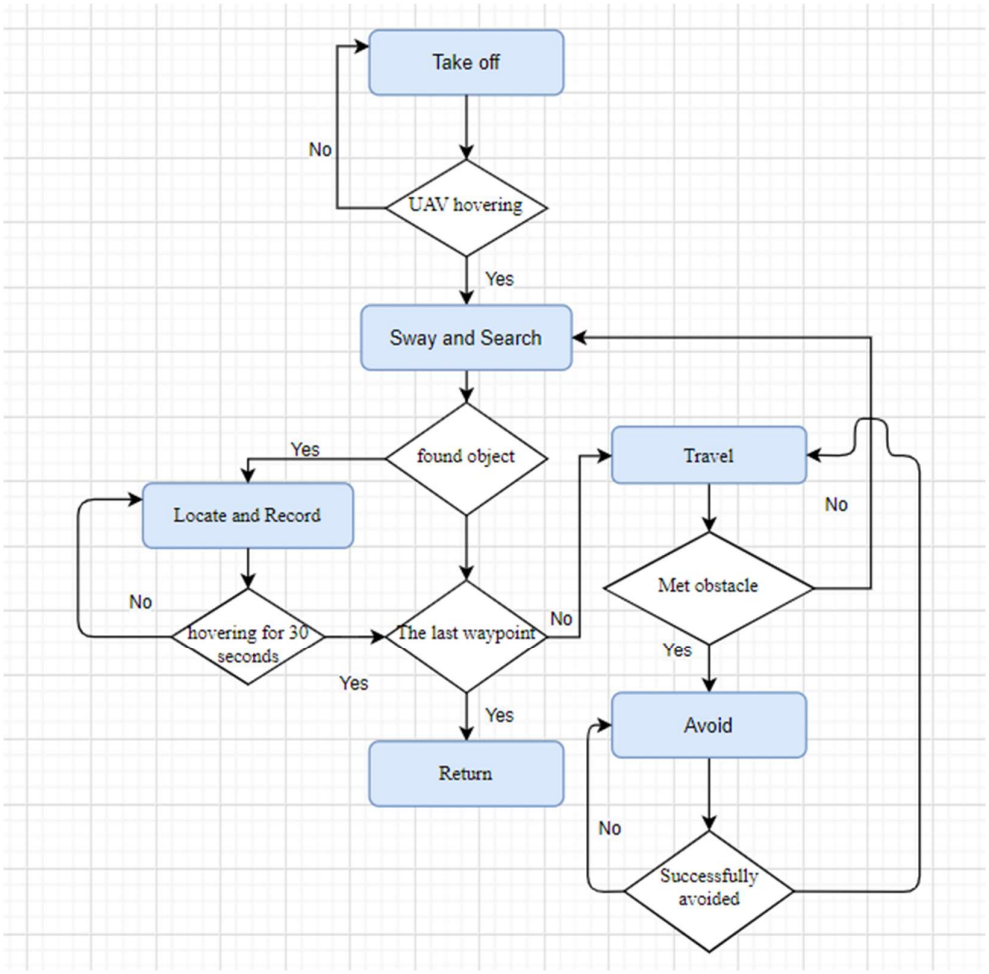


Fig. 4.4 FSM of object searching

5. Methodology – Object Tracking

As mentioned in Section 2.2, a consistent object tracking by UAV requires perception capability (i.e., object detection and object motion prediction) and the backend actions of UAV. Hence, our object tracking module could be discussed into two works: perception and backend actions. The system is designed to be capable of detecting, tracking, and further following the movement of a single tracked target object. The main objective is to enable one quadrotor UAV to follow a single object. Such application, for instance, this UAV system could be potentially helpful in assisting SAR missions or reconnaissance tasks. And thus, multiple-objects tracking problem will be temporarily disregarded at this stage. Obstacle-avoidance ability of UAV system during tracking mission is also disabled as we intend to examine how the system plan and act in accordance with target object's movement.

The structure of this part is that, Section 5.1 will firstly discuss our perception solutions, which are YOLOv4-Tiny and Kalman Filter Estimation, whereas Section 5.2 will describe the tracking-control actions of the UAV designed by our team.

5.1 Object Tracking with Perception

5.1.1 Object Tracking with YOLOv4-Tiny

Similar to the elucidated object searching system in Section 4, the foundation for performing object tracking is a satisfactory perception for UAV vision-based system to recognize the to-be-tracked object. A real-time object detection algorithm is the prerequisite for tracking the object across multiple images. Therefore, same as the object searching module, we exploited YOLOv4-Tiny as our 2D detector due to its good trade-off between speed and accuracy. The rationales of adopting YOLOv4-Tiny was explicitly justified in Section 2.1 and Section 4.1. The procedures in training and integrating YOLOv4-Tiny have no differences with the abovementioned operating details; so, we refer the readers to Section 4.1.

However, YOLOv4-Tiny framework could not completely resolve the needs of perception, as the UAV system still demands an algorithm to predict the motion of objects. Besides, on account of the camera's self-motion during flying and the occlusion problem of the tracked object, the object detection performance of YOLOv4-Tiny is likely to be degraded. To resolve such issue, we further applied a Kalman Filter to keep predicting the object location, from the

moment that UAV loses track of the object to the time that the object reappears. The thorough methodology of motion prediction with Kalman Filter is given in next chapter.

5.1.2 Object Tracking with Kalman Filter (KF)

5.1.2 (a) Discrete Kalman Filter

The famous Kalman Filter was firstly co-invented by Hungarian engineer Rudolf Kalman in 1960 and was applied in the navigation system of Apollo mission in the late 1960s (Welch, 2009 & Humphreys et al., 2012). The Kalman filter, to simply put, is a mathematical model that aims to deal with a dynamic system possessing uncertain state information and to attempt to make educated estimation for the system in next time step. It works iteratively and is considered to be suitable for systems that are continuously changing. It also usually appears in applications where several sensors (i.e., data fusion, e.g., GPS and IMU) act as the input and require an optimized estimation of the state. The following part illustrates the main components of the Kalman filter that was applied in our system.

The Kalman filter could first be understood as (Welch & Bishop, 1995 &Thacker &Lacey, 1998):

$$\hat{X}_k = K_k \cdot Z_k + (1 - K_k) \cdot \hat{X}_{k-1}$$

where the $\hat{X}_k$ is the current state estimation, K_k is the Kalman gain, Z_k is the measured state and $\hat{X}_{k-1}$ being the state estimation at previous time step. From above, the filter basically considers the previous state, the newly measured input and the averaging factor (i.e.,K_k) and then predicts the state estimation. Particularly, K_k, the optimal averaging parameter, is deemed as the most crucial value in the equation, as it affects the estimation result extensively. The following shows further derivation process. By assuming all variables in state vectors to be Gaussian (normal) distributed, the state $\hat{x}_k$ and measurement z_k are expressed in:

$$\hat{x}_k = Ax_{k-1} + Bu_k + w_{k-1}$$
$$z_k = Hx_k + v_k$$

The state $\hat{x}_k$ are assumed to be a linear sum of the previous state values, the input control signal u_k, as well as the process noise w_{k-1} and the measurement noise v_k. As for A and B, they are

the transition matrices between time step k and k-1, whereas H is the noiseless connection matrix of state and measurement. The model can thus be discussed into 2 steps: Time update (prediction) and Measurement update (correction).

Time update (prediction):

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1}$$
$$P_k^- = AP_{K-1}A^T + Q$$

Measurement update (correction):

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$$
$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$
$$P_k = (1 - K_k H)P_k^-$$

Specifically,

$$Q = E[w_k\, w_k^T]$$
$$R = E[v_k\, v_k^T]$$

The two matrices (Q and R) are the covariance matrices of noises (w_k and v_k), and P_k is the error covariance matrix.

The main objective from the above equations is to obtain $\hat{x}_k$ at every time step k. Yet, to acquire the optimal $\hat{x}_k$, the optimal Kalman gain K_k is required. And to get the optimal K_k, matrices of $P_k^-, H, \& R$ are needed. Yet, for the acquisition of those matrices, the matrices at previous time step are further required. Therefore, the two steps, as well as the matrices, are counter-related, and they work iteratively in a loop. Fig. 5.1.2 below demonstrates the whole process:

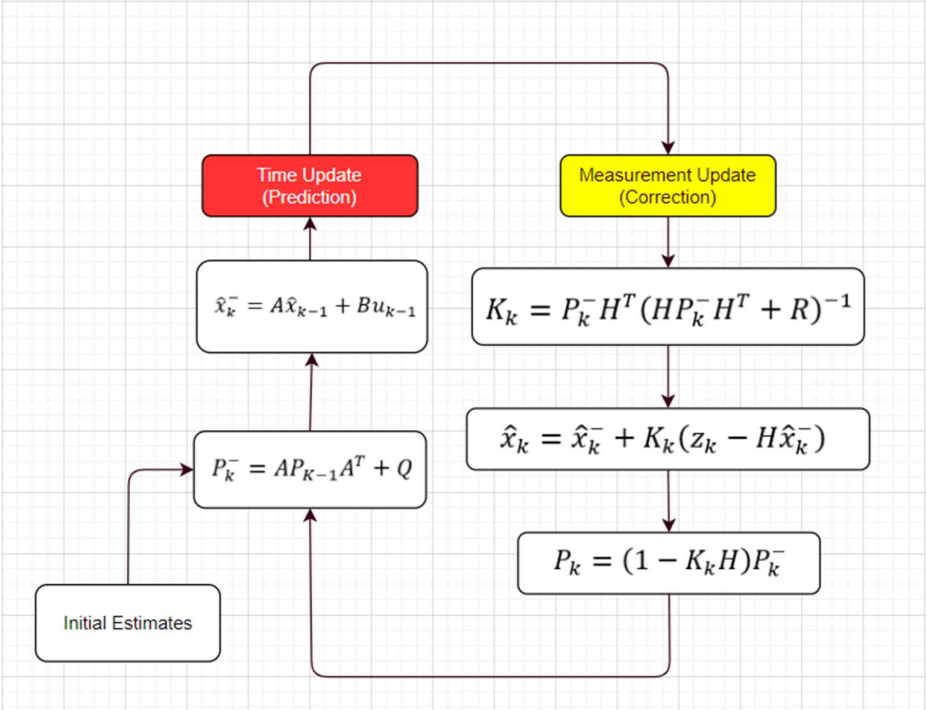


Fig. 5.1.2 the Kalman filter cycle (Welch & Bishop, 1995)

The priori estimates, which are denoted as $\hat{x}_k^-$ and P_k^-, act as the predicted values in prediction stage. The two are derived from either the initial estimates or the estimates at previous time step k-1. After getting the priori estimates, the prediction step will project them to next step for measurement update. The K_k will then be derived and subsequently, based on the input from measurement, the model will 'correct' the estimation of $\hat{x}_k^-$ and generate $\hat{x}_k$. In addition to that, the error covariance matrix P_k will also be updated based upon K_k and the priori estimate P_k^-. The two, also called posteriori estimates, will be fed to time update step for next priori estimates at prediction steps. The process will continuously iterate and continuously output the optimal estimation of states.

The Kalman filter is deemed to be robust in performing in linear systems, yet, for systems that act nonlinearly, the Gaussian characteristics might disappear, and hence the varieties of Kalman filter, such as Extended Kalman filter (EKF) and Unscented Kalman filter (UKF) (Ribeiro, 2004; Wan & Van Der Merwe, 2000), were proposed to deal with nonlinear system. Yet, as this project simply assumed that the observed object travel linearly, the details of EKF and UKF will not be further discussed. For more information on Kalman filter, we refer readers

to Welch and Bishop's (1995) introduction to the concept, in which the crucial concepts were discussed.

5.1.2 (b) Tracking Algorithm

Section 5.1.2-2 will thus present the implementation of Kalman filter in our tracking front-end work as well as the overall tracking algorithm.

By assuming the linear system, we applied the discrete Kalman filter in our tracking system, where the state vector column is:

$$\hat{x}_k = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}_{6\times 1},$$

$$while\ u\ is\ neglected\ as\ we\ have\ no\ input\ information\ of\ the\ object$$

The variables of the state vector are respectively the camera frames of the x, y, z and the velocities of the x, y, z, where, specifically, x and y are the coordinates on the image plane while z is the depth (i.e., the distance between the robot and the object). Therefore, the states will be continuously measured, estimated and updated, in which the measurement $\hat{z}_k$ will be input by YOLO-Tiny object detection algorithm, and the priori estimate $\hat{x}_k^-$ and posteriori estimate $\hat{x}_k$ will be constantly predicated and updated.

In particular, when $\hat{z}_k$ is not being measured, the system will conduct prediction based on the previous states. Among the covariance matrices, including noise matrices and error matrices, Q, R and P_k, P_k^-, both Q and R are predefined, whereas P_k^- is initialized at first and later being estimated at every time step. P_k, similar to $\hat{x}_k$, will also be updated at update step of Kalman filter. Furthermore, as the main function of the Kalman filter is to provide guidance information when the object is occluded or overlooked by YOLO-Tiny, the measured values from YOLO-Tiny will be used as main considerations for UAV path planning given that the object were successfully detected by the deep learning algorithm. Yet, in contrast, when YOLO-Tiny could not locate the object, the robot will take the estimates from Kalman filter for references. The

predicted states will act as the main object pose information until YOLO-Tiny detects the object again and updates the state.

Algorithm 5 concludes the workflow of our tracking method.

Algorithm 5 tracking algorithm

Input: image *F*
while *true* **do**
    object detection on F
    if object detected **then**
        trigger and initiate Kalman filter (KF)
        break
    else
        continue
    end if
end while

while *true* **do**
    object detection
    prediction step in KF
    if object detected **then**
        set update z_k with the observation state input
    else
        Set update z_k with the previous state
    end if
    conduct update step in KF
    Output: $\hat{x}_k$ (posteriori estimate)
    continue
end while

5.2 UAV Back-end Actions

5.2.1 UAV Path Planning during tracking

After completing the perception module work in section 5.1, we further defined the following action of the drone. To allow the object to be within the FoV of the quadrotor, it is considered

that the optical axis should be continuously focusing on the object. In addition, to avoid collision with the object, a safety distance D_{safe} between the UAV and the object should be maintained. Nevertheless, as the system is designed to follow a certain object, a maximum distance D_{max} should not be excessed. By designing the robot to face continuously towards the object as well as maintaining a distance, it is considered that the robot will be able to track and follow one single object.

To achieve the above, we simply divided the aerial image into equal 9 regions, as shown below in Fig. 5.2.1; in particular, the middle region is denoted as S_c.

Fig. 5.2.1

In addition, with the estimated or detected states derived from section 5.1, we defined:

$$O_c = (x, y)$$
$$d = z$$

From above, the system will then try to let O_c, which is the centre of the object, be within S_c while keeping d, the distance between UAV and object within D_{safe} and D_{max}. It is achieved, in an efficient and simple fashion, through adjustment of the robot's yaw angle and generation of waypoints for UAV to follow. For instance, whenever the object starts to gradually move away from the robot, the UAV will try to move closer. In addition, the UAV will be able to

modify its attitude when the object travels relatively to the right or left of the robot; the UAV will also move up and down if the object is either increasing or decreasing its altitude.

Algorithm 6 further shows and concludes the path planning module for the object tracking system.

Algorithm 6 tracking path planning

Remark: ψ, z_w are the pose variable of the UAV, and $WP_c\,(camera\ frame)$ and $WP_w(world\ frame)$ are the waypoints for UAV to travel to.

1. **while** *true* **do**
2. **if** $O_c.x < S_c.x_{min}$
3. $\psi = \psi + \Delta\psi$
4. **else if** $O_c.x > S_c.x_{max}$
5. $\psi = \psi - \Delta\psi$
6. **else**
7. $\psi = \psi$
8. **end if**
9. **if** $O_c.y < S_c.y_{min}$
10. $z_w = z_w - \Delta z_w$
11. **else if** $O_c.y > S_c.y_{max}$
12. $z_w = z_w + \Delta z_w$
13. **else**
14. $z_w = z_w$
15. **end if**
16. **if** $d < D_{safe}$
17. Set $WP_c = \{0,0, d - D_{safe}\}$
18. $WP_w = T_C^W WP_c$
19. **else if** $d > D_{max}$
20. Set $WP_c = \{0,0, d - D_{max}\}$
21. $WP_w = T_C^W WP_c$
22. **else**
23. $WP_w = WP_w$
24. **end if**
25. **end while**

5.2.2 Finite State Machine (FSM)

Finite State Machine (FSM) also guides the actions of our UAV during the object tracking module. In this module, the FSM consists of four states: take off, sway and search, follow, land. Fig. 5.2.2 depicts FSM with the states and the inputs triggering each state transition for the object tracking module. The following paragraphs explain each of the aforementioned states in more detail.

Take off: The take-off state is the same as that of Section 4.4.

Sway and Search: The UAV maintains its altitude and performs yaw motion in 360 degrees. At the same time, the onboard stereo camera will detect objects that are pre-trained in our custom model. If the target object is detected, the FSM will transit into *Follow* state. Otherwise, if the UAV cannot detect any object after a pre-set time period, the FSM will be switched to *Land* state.

Follow: The UAV will follow the object movement continuously, maintaining a safety distance while updating measurements (i.e., bounding boxes information and depth measurement) at every time epoch. If no new measurement update is received for a certain amount of time, the FSM will enter *Land* state.

Land: The UAV is commanded to land at its current ground location slowly at a constant vertical speed.

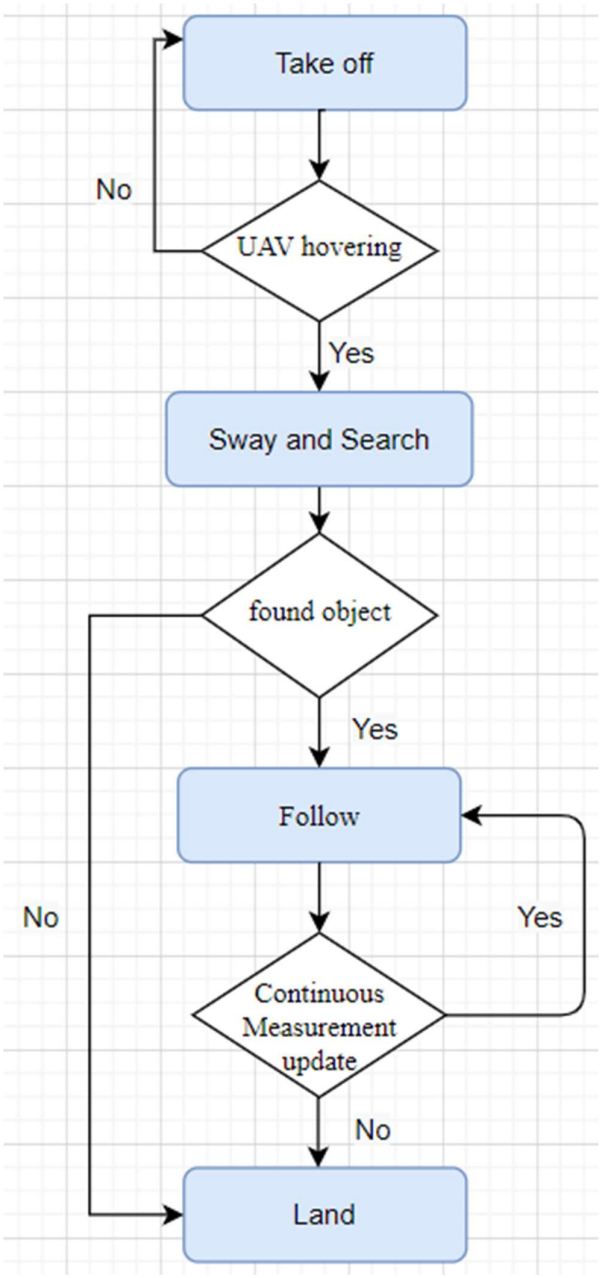


Fig. 5.2.2 FSM of object tracking

6. Experiments Results and Discussions

To validate the utility and effectiveness of the proposed UAV navigation system in performing object searching and object tracking in this paper, numerous experiments in both simulated environment (Gazebo simulator) and real-world scenarios under VICON arena were conducted. We tested the methodologies by using UAV system described in Section 3. The main content of this section presents the experimental setups, experimental results, as well as the analysis of results.

In brief, the objectives of designing the listed experiments are summarized as follows:

(1) Object detection module

- Section 6.1, the training result of our Yolov4-Tiny model on custom dataset demonstrates the performance of object detection deployed in embedded UAV system.

(2) Object Searching module

- Section 6.2.1, the flight test result under the Gazebo simulator proves the feasibility of our obstacle-avoidance algorithm in autonomous flight
- Section 6.2.2, after the successful evaluation of both object detection and UAV obstacles-avoidance, the experimental results of object searching flight under the VICON area as well as Gazebo validates the robustness of our UAV system in locating the position of target object in an unknown environment.

(3) Object Tracking module

- Section 6.3.1, the observation result assesses the real-time tracking performance on image plane in terms of 'false attempts' and occlusion before embed in the UAV system.
- Section 6.3.2, beyond the accomplishment of acceptable tracking on real-time images, we applied the system onto the UAV robot. The experimental results of object tracking flight in real-world VICON stage verifies the integrity and effectiveness of entire UAV navigation system in tracking object.

6.1 Training Result of YOLOv4-Tiny Object Detector

The quality of using 'YOLO' framework in operating real-time object detection as well as 3D pose estimation greatly relies on the training result of the model on our custom dataset. In other

words, the training result of object detector directly affects the performance of our UAV system in object searching as well as object tracking. Therefore, we need to conduct experiments on examining the detection performance before all else. This section specifically shows the result of the training model in our UAV system and further discusses the outcome.

6.1.1 Experiment-1: Comparison of performance between YOLOv4-Tiny and YOLOv4

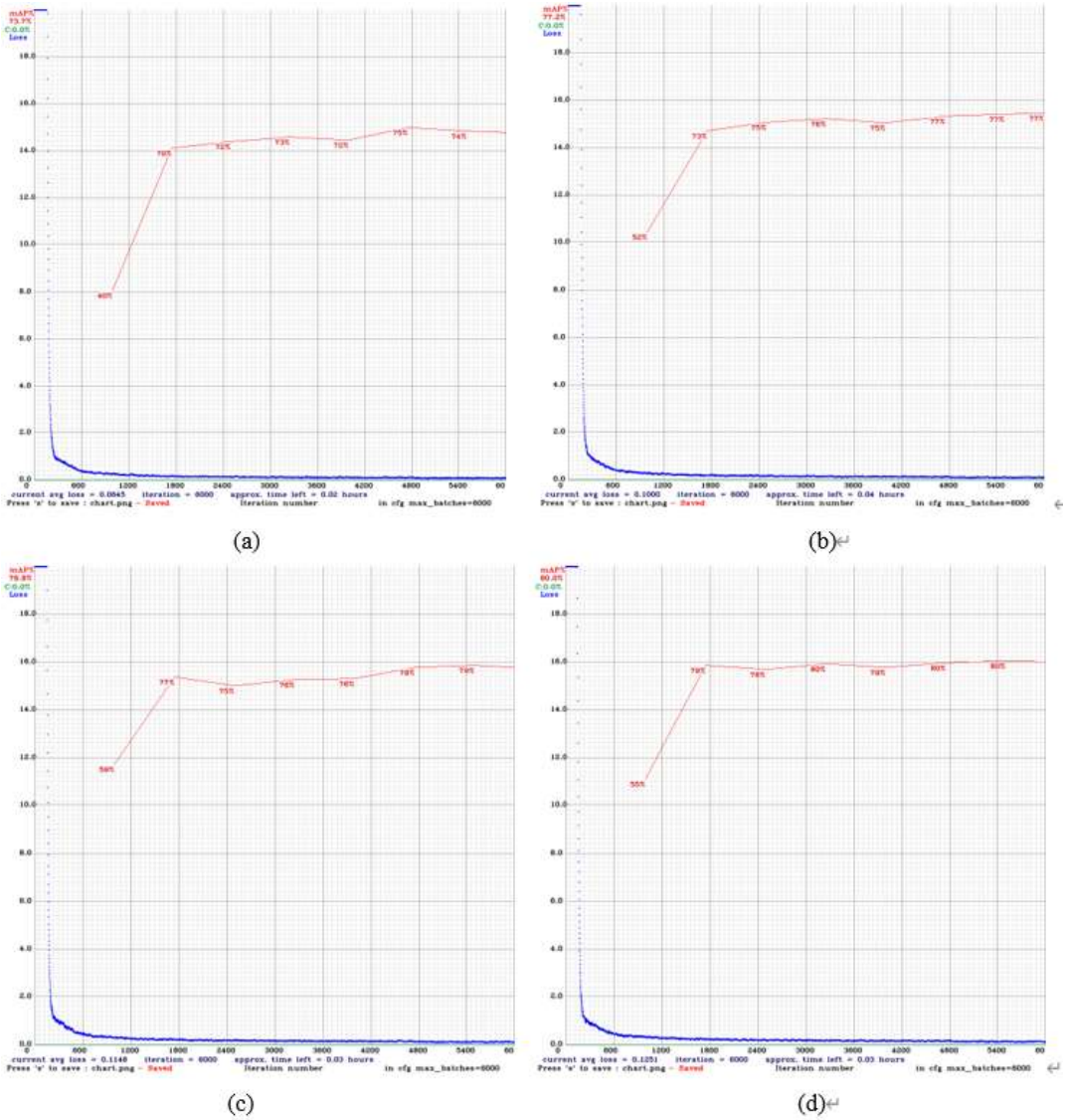
In the first experiment, we first made comparison of performance between YOLOv4-Tiny and YOLOv4 framework in the same size. Based on the result, we further confirmed the preference of YOLOv4-Tiny over YOLOv4 as the highest priority of performing both object searching and object tracking is the detection speed, the number of frames per second (FPS). In real-time autopilot operation, UAV needs to acquire the potentially greatest perception in every second in order to process navigation strategic. Low frame rates would cause serious delay of response of UAV's actions, especially when our vision-based UAV system predominantly banks on perception solution to address problem of object positioning, obstacles-avoidance, object tracking. For instance, if the object detector of UAV processes extremely low FPS, it is almost impossible to consistently track a moving object across a series of frames, not to mention the unavoidable motion blur problem on real-time aerial image. Oktay et al. (2018) and Liu et al. (2020) both admitted that the motion blur triggered by the inevitable vibration of the UAV during manoeuvre could significantly deteriorate the performance of UAV in real-time application; thereby we need to consider this factor in choosing the model. The comparison between YOLOv4-Tiny and YOLOv4 is concluded in the Table 6.1.1.

6.1.2 Experiment-2: Comparison of performance in different input resolution

Meanwhile, our custom model was trained to detect three different classes (i.e., Yellow Bulb, Pooh, Human), and each training process lasted 6000 iterations at which the training loss did not decline any further. Since different network resolutions could influence the model precision (Bochkovskiy, 2019), in second experiment, we trained our model with different resolutions (e.g., 320×320, 416×416, 512×512, 608×608) in YOLOv4-Tiny to evaluate the best model performance. The training performance curve of each input size are depicted in Fig.6.1.2. The comparison of the four input resolutions in terms of accuracy (mAP) and frame per second (FPS) is also demonstrated in Table. 6.1.1.

Method	Backbone	Size	mAP@0.50 (AP_{50})	FPS (@TX2)
YOLOv4-tiny	CSPDarknet-53-tiny	(a) 320 × 320	74.85%	5.2539
		(b) 416 × 416	77.21%	5.3657
		(c) 512 × 512	79.36%	5.5503
		(d) 608 × 608	80.20%	5.1557
YOLOv4	CSPDarknet-53	416 × 416	97.09%	1.0521

Table. 6.1.1 Demonstration of the performances of YOLOv4-Tiny and YOLOv4 with respect to different resolutions.



(a) (b)

(c) (d)

Fig.6.1.2 The relation of mAP and average loss to the iteration number for different input resolutions: (a) 320 × 320 (b) 416 × 416 (c) 512 × 512 (d) 608 × 608

Based on our observation of second experiment, the following discussions are made:

(1) Notably, larger input resolutions will increase the best possible mAP, but will inevitably slow down training process and the inference speed. Thus, it is not necessary to train higher input resolution as we achieved the acceptable speed and accuracy at 608 × 608 to fulfil our needs.
(2) Besides, as we deployed YOLOv4-Tiny on embedded computational- resource-limited onboard GPU (Jetson TX2), the achieved FPS was reasonably lower than the normal performance on offline computer GPU. We found out that it was extremely hard to achieve higher FPS above 10 even in YOLOv4-Tiny. For object searching task, the detector model of 5 FPS is sufficient as our UAV rotated stably at the same altitude to search items. For object tracking task, the performance with 5 FPS detector model will be further accessed and described in Section 6.3.1.
(3) When the input resolution was 608 × 608, our YOLO-tiny model performed best on the validation dataset at an 80.20 % mAP with intersection of union threshold of 0.50 (AP_{50}); but the frame per second (FPS) is slightly lower than that of input resolution 512 × 512. Since the mAPs do not differ significantly for these two resolutions and a high FPS is more critical to our real-time object detection task, we therefore chose the YOLOv4-Tiny model with a resolution of 512 × 512 as a good balance between detection accuracy and inference speed.

6.1.3 Experiment-3: Testing on real-time object detection performance

Once the training was completed, the third experiment on accessing the classification accuracy of detecting target objects on real-time videos captured on Intel RealSense D435i stereo camera was conducted. The result is displayed in Fig. 6.1.3 below. After we assured the validity of the performance of object detection, we could progress to further experiments for evaluating object searching and object tracking in next sections.

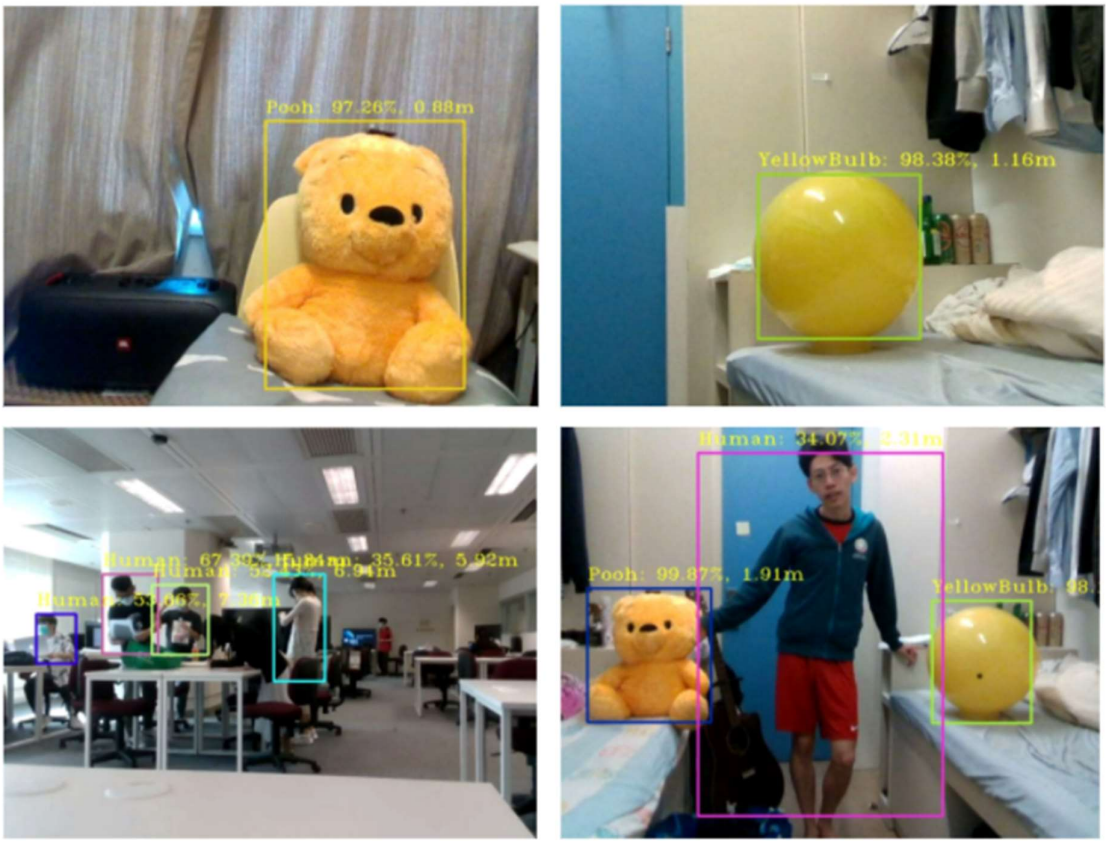


Fig. 6.1.3 Real-time detection results

Generally, there were low False Positives and low False Negatives found in the detection results especially the detection performance of 'pooh' shows the greatest result. However, the performance on 'human' class is relatively poor with low confidence score, and sometimes False Negative were occurred as shown in the third picture (the man in red shirt at back of the room and the girl in black dressing was not detected as human object). We figure out that it might be owing to the incomprehensive and low diverse training set for 'human' class, because we prepared a main large number of our team members' photo, and the majority of the captured backgrounds were either at the lab of AAE or the campus of HKPU. To improve such issue, the establishment of training set for 'human' class is suggested to be pluralistic in terms of background venue, body poses, dressing, and more variations. Human detection is widely known as the most difficult challenge in the field of computer vision. By means of 'trial and error', the training result could probably be further enhanced.

6.2 Object Searching

6.2.1 Experiment-4: Obstacle Avoidance Ability Test Result in Gazebo Simulator

To validate the robustness of Edge Detector Planner, we had tested the algorithm in Gazebo simulation (SITL). Based on the end-to-end simulation kit designed by Chen et al. (2021), we modified the world file of Gazebo map and ran the simulation on a modified Iris-quadrotor, which is equipped with a virtual Intel RealSense D435i RGB-D camera whose image resolution is 640 × 360. The UAV model is shown as below in Fig. 6.2.1 (a) (Chen et al.):

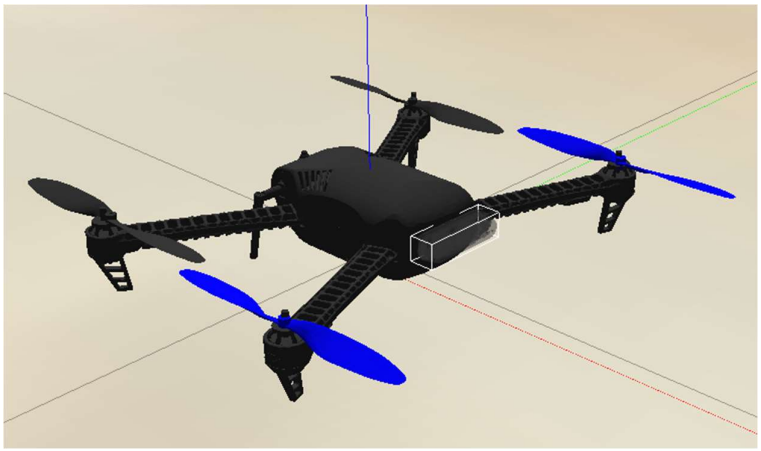


Fig. 6.2.1 (a) Iris Quadrotor embedded with Intel RealSense D435i

To test our Edge Detector Planner, we designed a map in which obstacles with different dimension were placed. Fig. 6.2.1 (a) and (b) show the custom-modified map in Gazebo:

Fig. 6.2.1 (a)

Fig. 6.2.1 (b)

As displayed in Fig. 6.2.1 (a), the UAV was originally sitting at origin of the 3D space. In this simulation experiment, the designed software would command the UAV to firstly take off to (0,0,1.2), then successively transverse to three different waypoints, which were respectively (14,0,1.2), (-10,7,1.2) and (-5,0,1.2). Hence, the Finite State Machine was set as:

- *Take-off*
- *Move-to-WP1*
- *Move-to-WP2*
- *Move-to-WP3*
- *Return*
- *Avoid*

After reaching all waypoints, the UAV would try to return to the initial point. In addition to the above, unknown obstacles were scattered throughout the path, whereby the UAV was expected to encounter 7 different obstacles. As our planner is based on Canny edge detection, different wallpapers with abundant visual features were attached onto the obstacles. Thus, the UAV would confront different obstacles with different sizes, as well as different features. In addition to those, below (Table 6.2.1 (a)) shows the set parameters. In particular, the safety radius R_{safe}

was set as 1.2 m, and since the simulation scenario was relatively spatial, the empirically pre-defined $d_{predifined}$ was placed as 1.6 m.

R_{safe}	1.2
$d_{predefined}$	1.6
v_{max}	0.5

Table 6.2.1 (a)

The figures (Fig. 6.2.1 (c) – (e)) below present the result of the experiment displayed in RVIZ:

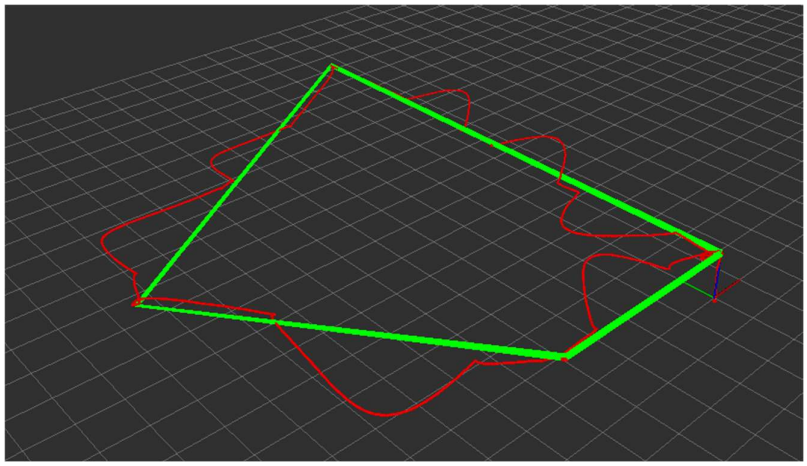


Fig. 6.2.1 (c) the final trajectory of the Gazebo simulation, where red curve shows the real trajectory of the simulation, while green shows the course if there exists no obstacle.

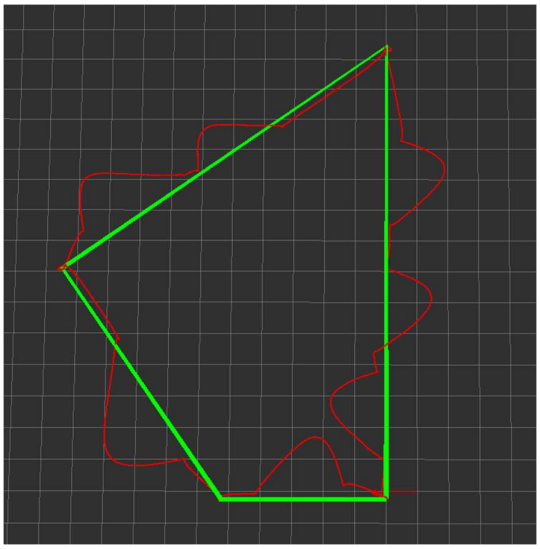


Fig. 6.2.1 (d) the bird view of above result

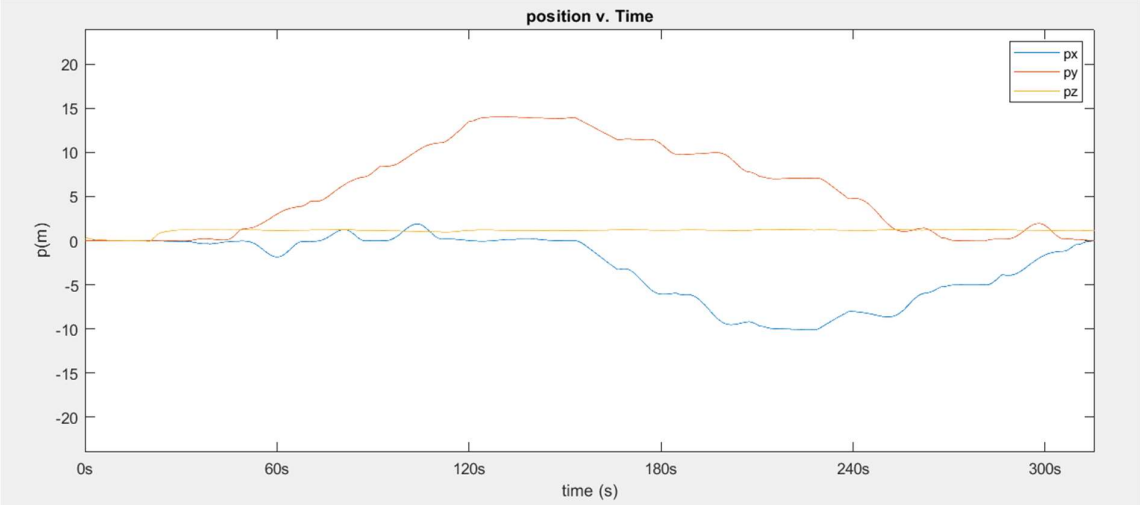


Fig. 6.2.1 (e) the position flight data of the UAV

From the simulation results, the planner showed that it could avoid obstacles with different dimensions, travelling safely and arriving at each waypoint. Fig. 6.2.1 (f) shows some of the first-person view of the quadrotor during the simulation test.

Fig. 6.2.1 (f) different obstacles that the UAV had encountered.

The top left is set to be 1 × 1 × 2.5, while top right also being 1 × 1 × 2.5 yet with a different encountering angle. As for the below 2 pictures, bottom left is set as 0.6 × 0.6 × 1.25 and bottom right is considered to be 1.4 × 0.4 × 2.0. The red dot is the derived reference point (for waypoint search) from the Edge Detector Planner

Another goal of the simulation test is to observe the dynamic performance of the UAV as well as evaluating the computational speed of our Edge Detector Planner. Fig. 6.2.1 (g) displays the velocity profile throughout the flight. As shown, the velocity was well-controlled, approximately, under 0.5 m/s. Yet, it could also be seen that the UAV were sometimes drastically accelerated, and this was due to the fact that only the trajectories during obstacle avoidance were optimized. Therefore, the velocities were sometimes smoothly increased or decreased, while sometimes abruptly. In addition, from the simulation and the afterward results, we noticed that such local planner could only work robustly for column-wise obstacle. Yet, when encountering walls, more complex obstacles or a dead-end of a corridor, the Edge Detector Planner is most likely to fail. We will further discuss the possible improvements in Section 7.2.

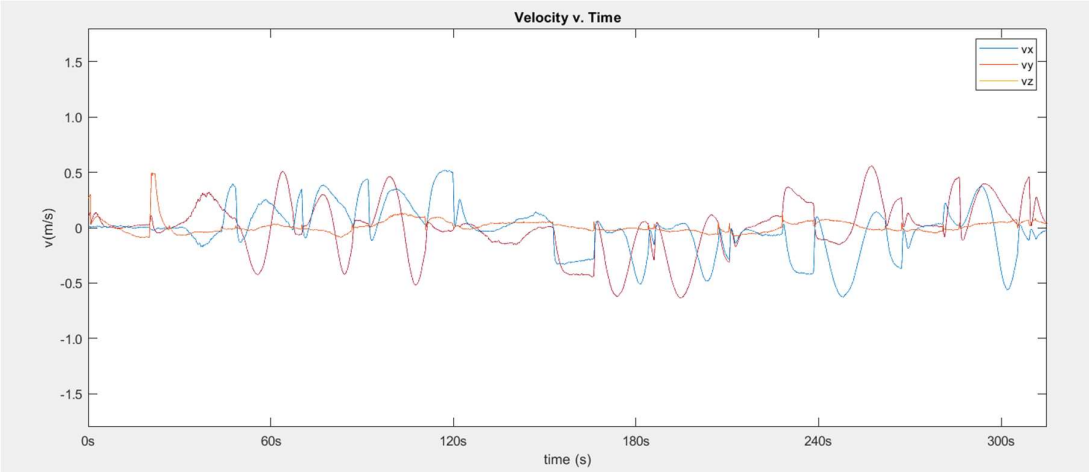


Fig. 6.2.1 (g)

As for computational speed, during the flight test, the computational time of average collision check was 35.30 ms, while the average computing time for finding a new path when encountering an obstacle was 36.20 ms. This implies that if an obstacle obstructs the UAV's main course, the robot will need a total computational time of 71.50 ms, which is deemed not significantly fast yet satisfactory (Chen & Lu, 2020). The root cause for such moderate computational speed is discerned to be the image processing time; yet, with our Edge Detector Planner working efficiently during the simulation, we still considered the low computational speed acceptable.

The video recordings for both simulation and version of UAV's field of view (FoV) are uploaded to YouTube platform. The links are attached at Section 10 (Appendix I).

6.2.2 Experiment-5: Object Searching Flight test under VICON and Gazebo

After consolidating the performance of both object detection and path planning module, we further merged them as an object searching system. The system, as mentioned at the beginning of Section 4, is able to enter a 3D space without knowing the environments and then to search for a target object. The system will further conduct pose estimation via frame transformation and record down the location of the target object. Therefore, we designed a scenario under the VICON system, in which the UAV would attempt to find the object; yet, due to space limitation, we could only setup a relatively simple mission for the experiment. As for the system architecture, it could be referred to Section 3. The input image from the RGB-D camera was with 640 × 480 pixels, which is different from the one in Section 6.2.1. The setup scene under the VICON is shown in Fig. 6.2.2 (a):

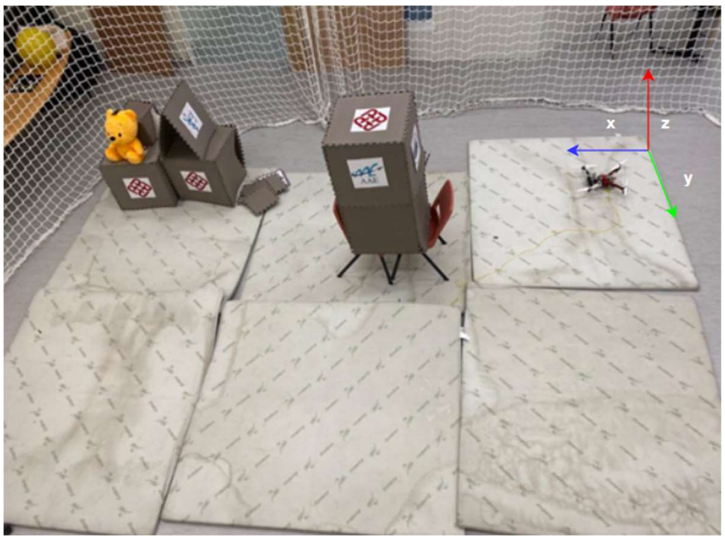


Fig. 6.2.2 (a) Setup of Flight Test in VICON platform

As displayed, the UAV was initially placed at origin, and a target object (the stuffed toy, which is classified as 'Pooh' in our custom trained model) sat at (4, 0, 0.6), approximately. The target object may not be visible for UAV after taking off to the first waypoint. After completing *Sway and Search* at the first waypoint, the UAV would be instructed to travel to next waypoint (4, 3, 1) and search for the target object by swaying. Yet, as observed, the UAV would probably

encounter an unexpected obstacle when moving to the waypoint. Thereby, the Edge Detector Planner would first allow the UAV to avoid encountered obstacle and then YOLO-Tiny would act as the object searcher. In particular, for Edge Detector Planner, since the scenario was relatively cluttered, the safety radius R_{safe} was set as 1.0 m, and the pre-defined $d_{predifined}$ was placed as 1.4 m. Additionally, the maximum velocity was set at $v_{max} = 0.5$. After reaching the waypoint, the UAV will start to sway, and if object found, the UAV would stop swaying and maintain its attitude, and further hover for 30 seconds while recording down the estimated pose of the object. After the object-localizing process, the UAV could return to home position or just land directly. During our experiment, for simplicity, the UAV just land directly. Hence, in our very experiment, the Finite State Machine was predefined as:

- *Take-off*
- *Sway and Search*
- *Move-to-WP*
- *Avoid*
- *Locate and Record*
- *Land*

Below (Fig. 6.2.2 (b) & (c)) show the final trajectory of the conducted flight test:

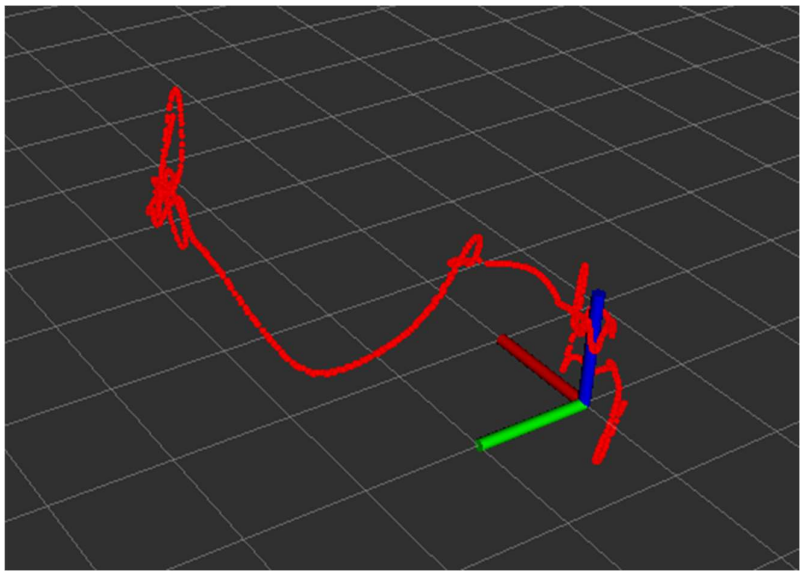


Fig. 6.2.2 (b) the trajectory of the flight test

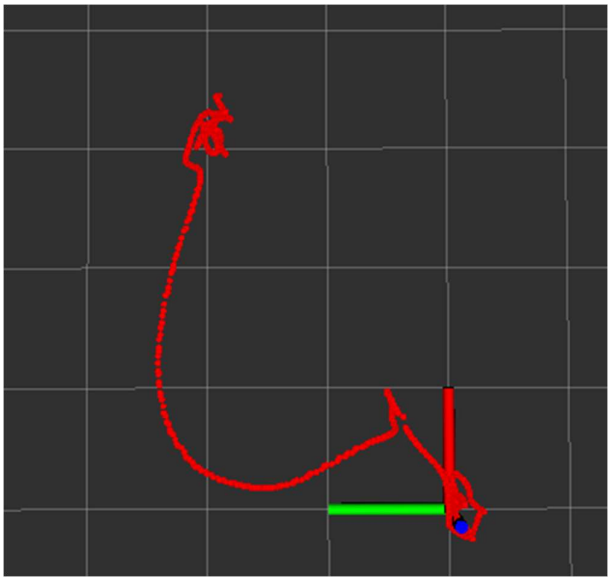


Fig. 6.2.2 (c) the bird view of flight test trajectory

During the flight test, our designed system successfully avoided the obstacle and found the target object. Below (Fig. 6.2.2 (d)) shows one of the acquired aerial images from the flight test:

Fig. 6.2.2 (d) acquired aerial image

Despite an obvious motion blur occurred in the image, our trained model was still able locate the classified object ‘Pooh’. Nevertheless, the image shown above was captured during the

hovering stage (where the robot should be relatively stable) of the flight test; by further referring to Fig. 6.2.2 (b) & (c), it could be observed that the robot severely jittered when taking off, hovering and swaying, hence occurred motion blur. To solve such problems, we had repeatedly calibrated the flight controller and the VICON system. Yet, after a considerable number of trials, the above-shown data were the most jittering-free results. Hence, we concluded that a technical issue took place during the experiment. The recorded pose information of the target object was then discarded due to the produced instabilities of the external system. It is considered that more flight tests will be conducted once the technical issue is fixed after the submission of this paper.

Notwithstanding the unevaluable acquired recording of object pose information, to further validate our pose estimation calculation, we setup a similar Gazebo simulation environment, with a similar dimension in VICON simulation, as shown below in Fig. 6.2.2 (e):

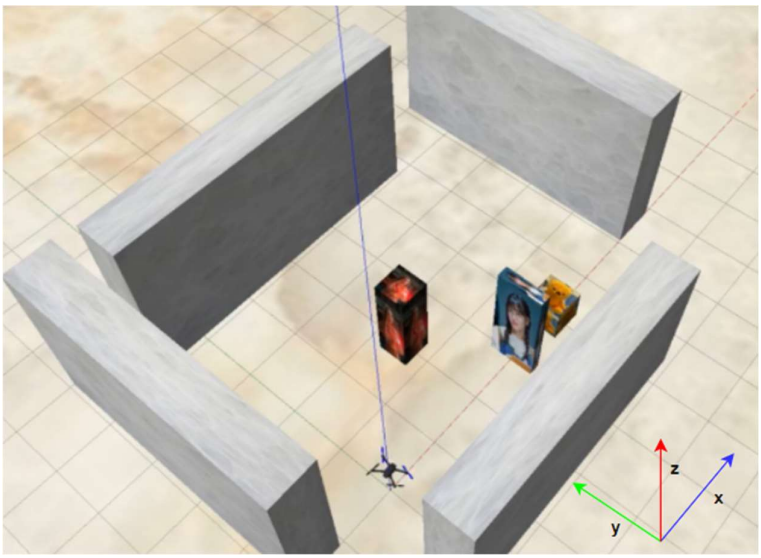


Fig. 6.2.2 (e)

Yet, for convenience, we did not design a *.sdf* model for the target object; instead, we simply attached the object image on a $0.6 \times 0.6 \times 0.6$ cubic, as shown in Fig. 6.2.2 (f).

Fig. 6.2.2 (f)

With the Gazebo simulation environment shown in Fig. 6.2.2 (e), we re-conducted the object searching flight test under the Gazebo environment and below (Fig. 6.2.2 (g) & (h)) shows the final trajectory of the simulation:

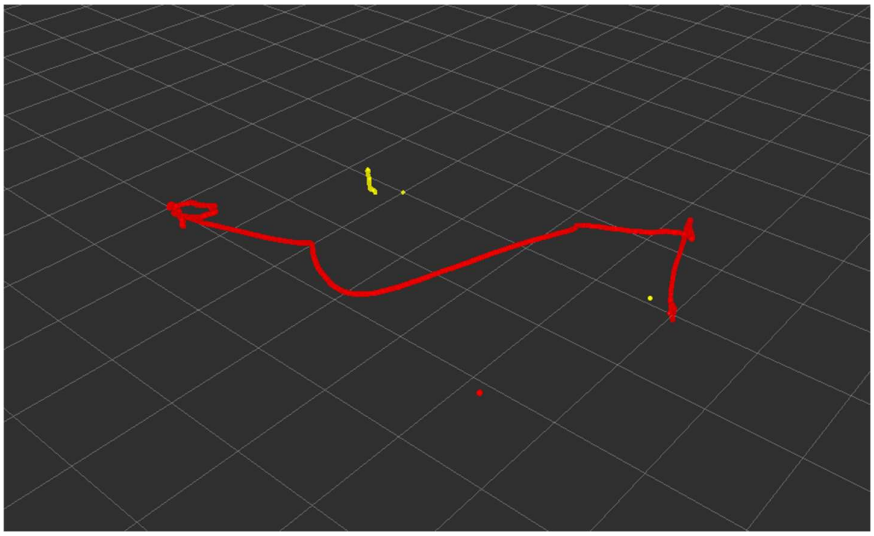


Fig. 6.2.2 (g) red curve shows the trajectory, whereas the yellow dots display the estimated pose of the target object

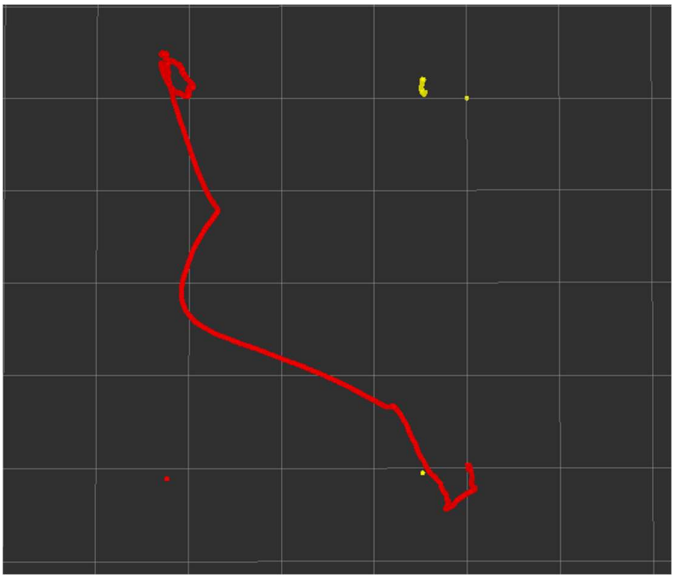


Fig. 6.2.2 (h) the bird view of the trajectory.

Compared with the trajectory from real flight test, this result seems to be jittering-free and much satisfactory. Additionally and more importantly, we could then evaluate the pose estimation method, in which an IIR filter was applied. During the *Locate and Record* state, the system had captured and detected 161 frames of aerial images, and further conducted pose estimation accordingly. The following series of plots are the results, and it shows that although there exists a certain amount of error between ground truth (straight lines in Fig. 6.2.2 (i), (j) & (k)) and estimation, the error could be mitigated by IIR filter as the values tended to converge at last. Nevertheless, when compared to the results presented by Feng et al. (2021), our estimation seems to be less accurate. The inaccuracies were ascribed to the relatively low accuracy of YOLO-tiny, in which the model localized the object on the image plane in a more unstable fashion than YOLOv4. This thus sometimes gave us a more fluctuating numerical result.

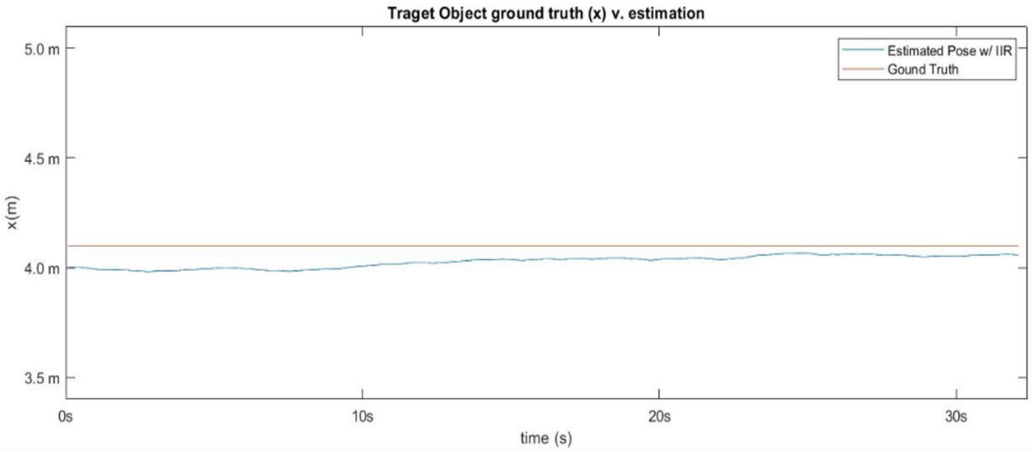


Fig. 6.2.2 (i)

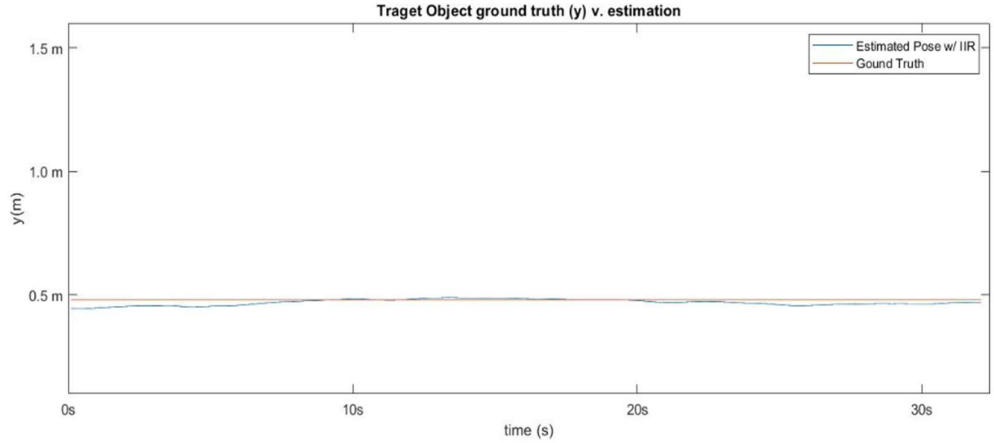


Fig. 6.2.2 (j)

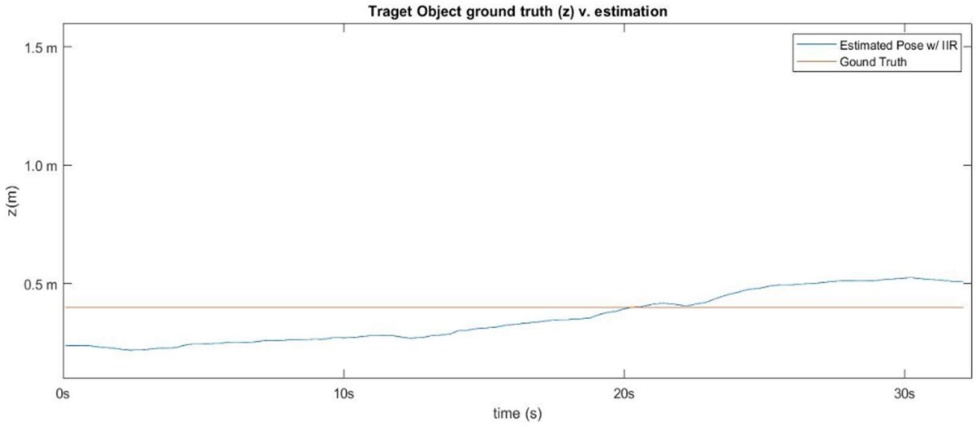


Fig. 6.2.2 (k)

Both video links of the flight tests under the VICON and the Gazebo simulator are attached in Section 10 (Appendix I). We emphatically encourage the reader to witness the full outcomes by reviewing the video footage.

6.3 Object Tracking

6.3.1 Experiment-6: Real-time Tracking Performance on Image Plane

Before we implement the object tracking system on the UAV device, we have to check the perception performance of our methodology (YOLOv4-Tiny Detection and Kalman Filtering Estimation) at the front-end stage with Intel RealSense D435i stereo camera. The high quality of YOLOv4-Tiny was proved in Section 6.1, but we still need to assess the performance of Kalman Filter in predicting the state of object trajectories across the captured frames. The experiment here was designed to evaluate the data association, prediction accuracy and the occlusion handling of our perception solutions (YOLOv4-Tiny and Kalman Filtering Estimation).

There were many general evaluation metrics in multiple object tracking (MOT) performance such as the conventional 'CLEAR' metrics (Bernardin & Stiefelhagen, 2008), Multiple Object Tracking Accuracy (MOTA) (Bernardin et al., 2006) and the latest 3D MOT evaluation tool (Weng et al., 2020). Nevertheless, they are not applicable to our case of single object tracking as our system will not face problems, such as the usual identity switches (IDS), associated with MOT.

For single object tracking, the 'false attempts' (Schlogl et al., 2004) and the occlusions (Lee et al., 2014) are perhaps more influential in system performance. 'False attempts' are defined as the false detection of object when there is no ground-truth object in the actual scene (Schlogl et al., 2004), and occlusions happen when a part of the object is screened by other objects or even background structure (Lee et al., 2014). In particular, we conducted real-time tracking experiments in video recording at the campus of the HKPU to further evaluate our system on these two criteria (false attempts and occlusions). We created the occlusion scenes on purpose to check how our system performed.

The following diagrams show some examples of occlusion scenes in order during the experiment. Note that the Kalman Filter (KF) will generate object state estimation represented by the red bounding box, which should be able to continually follow the motion of object in

positive results; YOLOv4-Tiny will produce object detection represented by the yellow bounding box, which should be capable to consistently recogniz the target object in positive results.

(1) Stationary object tracking- short-duration abrupt inter-object occlusion in outdoor environment

Diagram 6.3.1(a) Short-duration occlusion

In this process, our team was holding the target object ‘pooh’ at rest while an abrupt passer-by trespassed the view of camera in outdoor environment. There was an inter-object occlusion (frame#2---#5) in which a part of the ‘pooh’ was hidden for a short duration of 1-3 seconds. Apparently, the system did not lose track of the ‘pooh’ or get interrupted by the passer-by. The red bounding box was consistently located on the ‘pooh’. The state estimation of KF was considered as sufficiently satisfactory performance in handling short period of occlusion.

(2) Stationary object tracking- long-duration disappearance in indoor environment

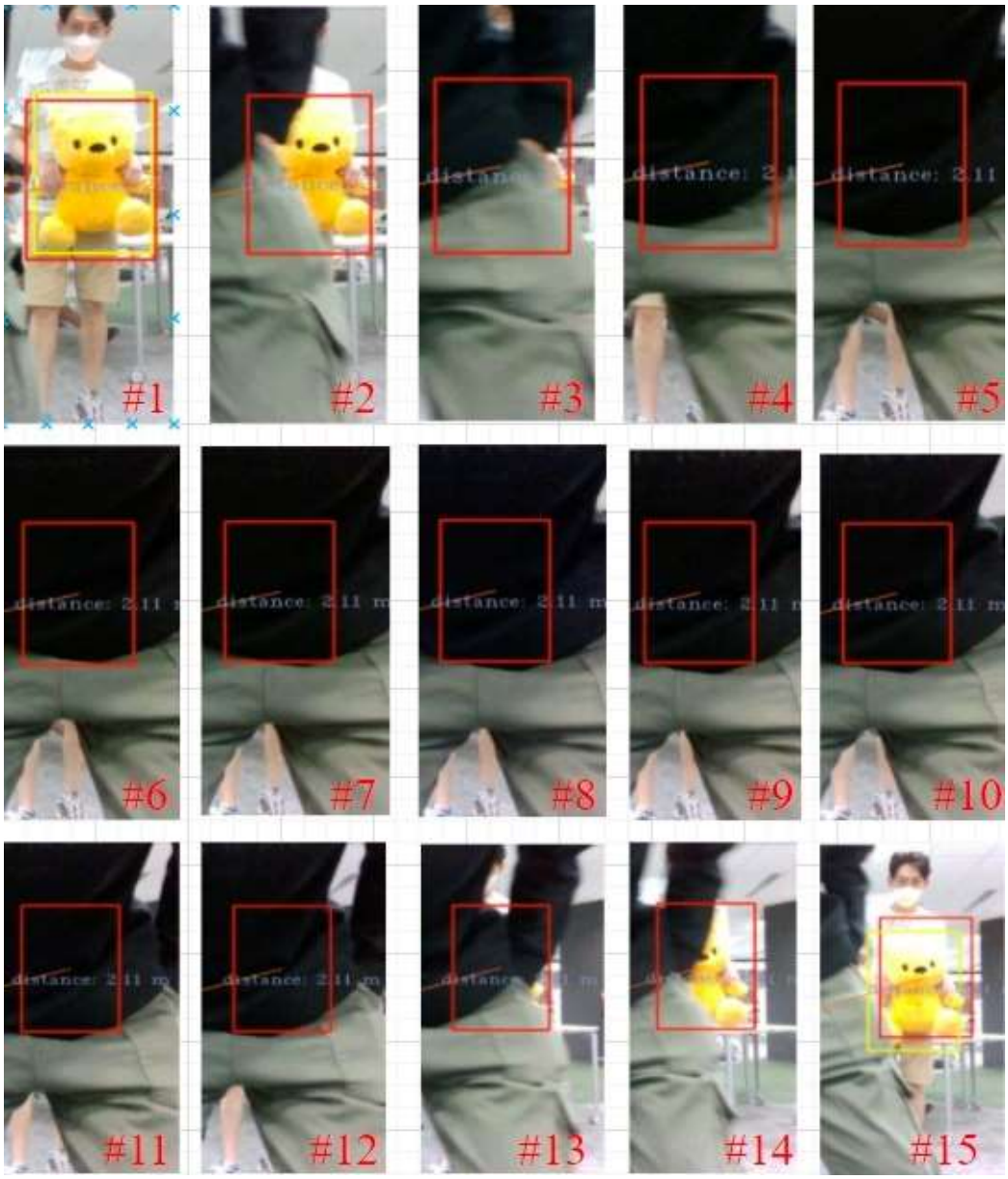


Diagram 6.3.1(b) Long-duration occlusion

Our team was tracking the unmoving target object ‘pooh’ in indoor environment while an unexpected thing blocked the camera view for a long duration of 5-8 seconds. Such long duration of object disappearance (frame #3---#13) required the KF to consistently predict the ‘pooh’ until the object reappeared. Based on the result, we could evidently observe that the red bounding box continuously predict the correct position of ‘pooh’ from calculating the previous frames. When a part of the ‘pooh’ reappeared, the system retrieved the tracking of ‘pooh’ immediately in frame #14. No false attempts were seen. When using UAV system in tracking object, we need to ensure that if YOLOv4-Tiny fails to provide KF with the position of object

in several seconds, the UAV needs to maintain high safety and integrity in autonomous flight by not triggering any uncontrollable moves due to the loss of object appearance. The result demonstrated that our designed system is able to track through longer periods of occlusion.

(3) Moving object tracking- ‘occlusion by the background scene structure’

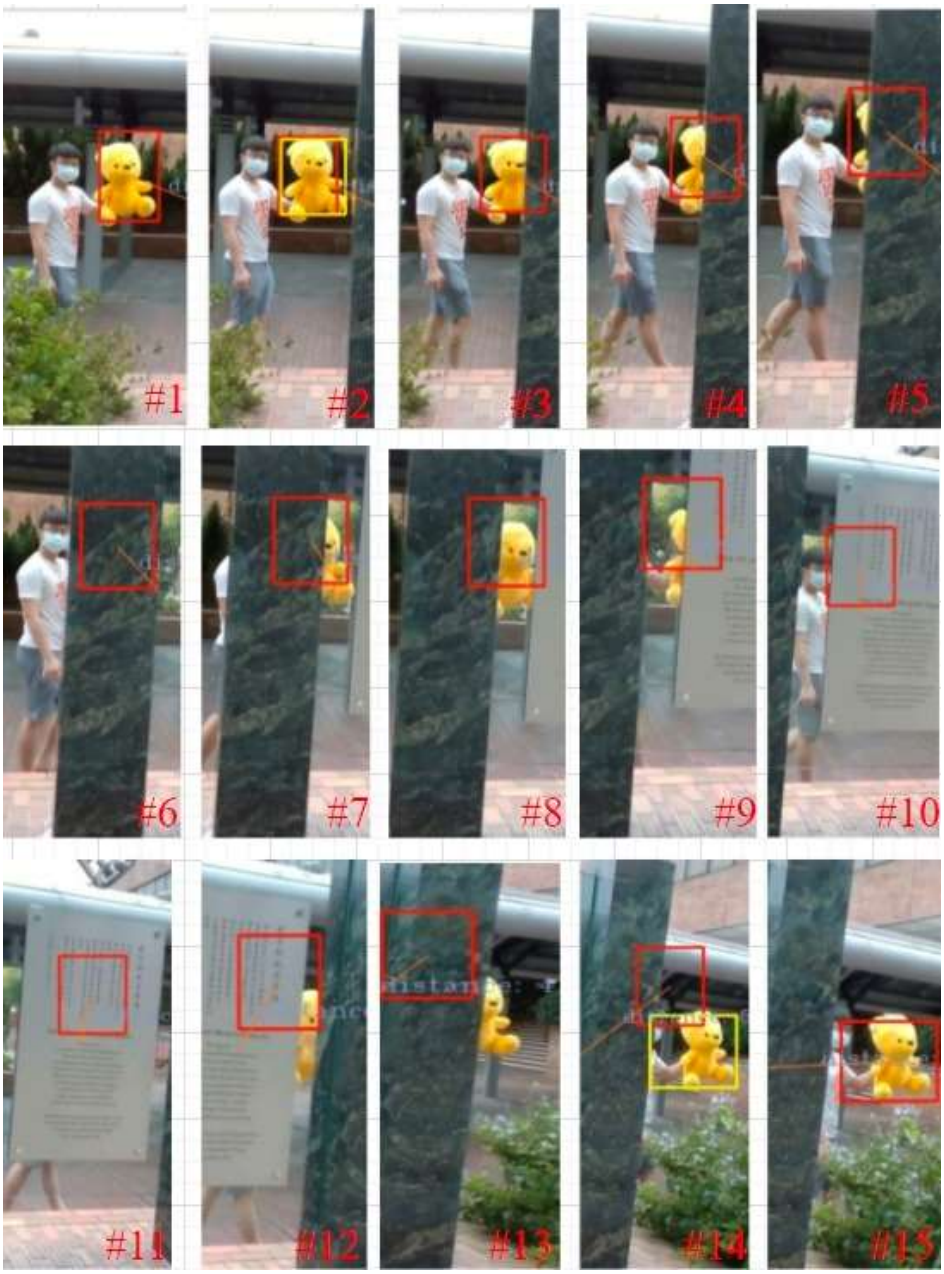


Diagram 6.3.1(c) Moving object tracking

Throughout these scenes, the target object 'Pooh' was moving from left to right and passing by a long memorial board. The system should stably predict the trajectory of moving 'pooh', based on its previous frames to the latest frame, in such occlusion caused by background scene structure. Based on the result, we could first notice that the red bounding box kept predicting the potential position of the 'pooh' even if it lost the plain view of 'pooh' from frame #4 to frame#11. It could roughly estimate the motion of the moving 'pooh'. However, we saw that there were some errors and deviations in frame#12---#14, in which the red bounding box did not precisely fall onto the 'pooh'. We deduced that it could be the reasons of the non-linear motion of the moving objects or the camera. This tracking deviation, although acceptable when object's motion is not aggressive, could be a direction for future improvement. This issue would be further discussed in Section 6.3.2 and also Section 7.3. Lastly, in frame #15, the system adjusted and restored the status immediately to re-track the 'pooh'.

We do not make additional qualitative and quantitative analysis in such experiment as our project focuses more on the implementation performance of UAV rather than the development of YOLO and KF in network structure. A complete object tracking system necessitates the UAV to coordinate the backend actions in response to the dynamic target object, and this is discussed in the next experiment, which is the real drone flight test.

In conclusion, the experimental result on perception stage is deemed to be a favorable outcome so as to move on to advanced experiment. The YouTube link recorded videos (i.e., Diagram 6.3.1(a)-(c) and additional trials) are included in the Section 10 (Appendix I).

6.3.2 Experiment-7: Object Tracking Flight Test Result under VICON

After confirming that our tracking method worked on real-time image planes, we further designed an experiment and tested the robustness of the system. The UAV, as explained in the beginning of Section 5, would be able to take off and search for a specific object, which was the class 'Pooh' in this case. After successfully locating the object, the UAV would infer its relative position to the object and further 'follow' it. The term 'follow' here simply means that the robot would try to adjust its optical axis towards the object while maintaining a certain distance. The finite state machine was then set to be identical to the states presented in Section 5.2.2. Fig. 6.3.2 (a) below first shows the setup of the experiment:

Fig. 6.3.2 (a) the setup of experiment – object tracking

As shown in the picture, the experiment was conducted under VICON. The UAV would be instructed to take-off to (0,0,1.4), and start swaying. If the object found, the UAV would start to follow the object. After the UAV successfully located the object and hover stably, we would enter the arena and manually move and carry the object around. In addition, to that, in order to create situations where the object is occluded, we intentionally trespass the space between the UAV and the object. As for the changing angle of yaw, we set $\Delta\psi = 0.08(rd)$, yet, for safety issues as well as the space limitation of the experiment occasion, we set a small ascending and descending value of $\Delta z_w = 0.005$ and relatively conservative parameters of $D_{safe} = 2.2$ and $D_{max} = 3.8$.

Fig. 6.3.2 (b) & (c) show one of the experiments' trajectories:

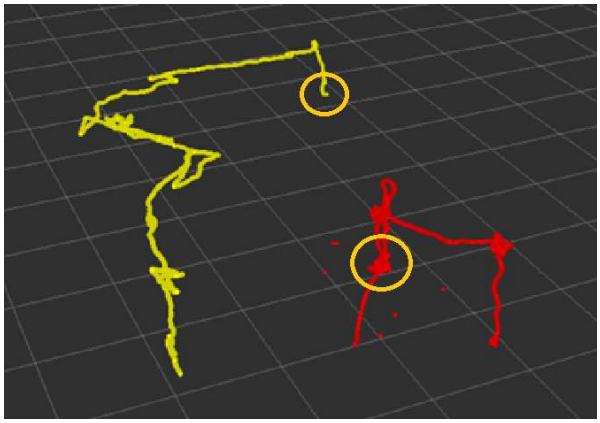


Fig. 6.3.2 (b) the trajectory of object tracking experiment. The yellow indicates the object's movement, while the red shows the UAV flying course. As for the orange circles, they show the time spot at which the UAV successfully located the object "Pooh".

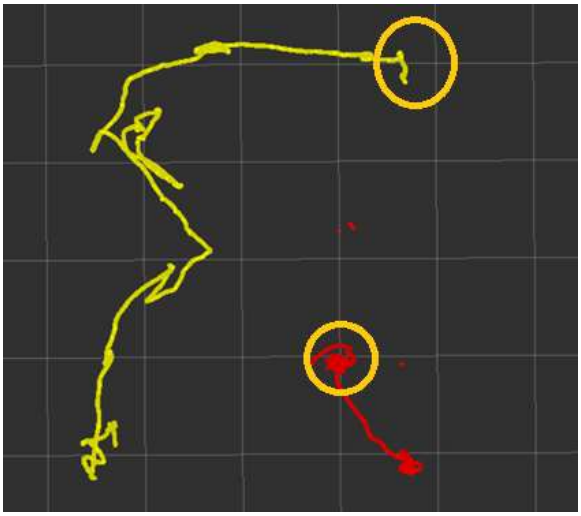


Fig. 6.3.2 (c) the bird view of the tracking experiment. The orange circles show the starting point of "follow stage".

From the observed trajectories, it could be seen that the UAV had successfully followed the motion of the object. Starting from the time point when the robot perceptively located the object, the UAV began to climb as the object climbed (Fig. 6.3.2 (d)). Afterwards, as the object moved horizontally, the UAV began to change its attitude, (i.e., yaw angle). The cluster of red dots shown in Fig. 6.3.2 (e) indicates the turning of the robot, and Fig. 6.3.2 (f) further shows the FoV of the robot during this period of tracking.

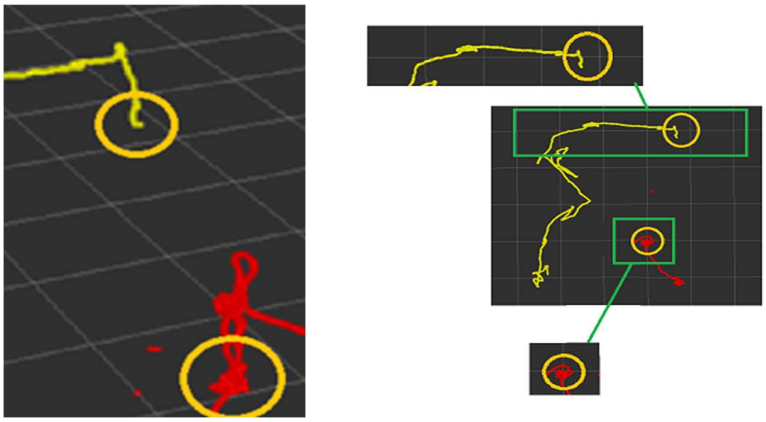


Fig. 6.3.2 (d) & (e) the zoomed-in version of trajectories

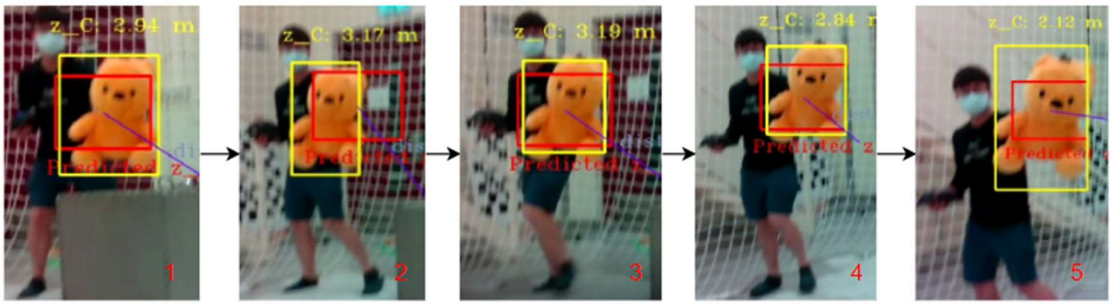


Fig. 6.3.2 (f) the FoV images of the UAV during tracking. The yellow bounding boxes were generated by YOLO-Tiny, whereas the red boxes are the object's state estimated by Kalman filter.

After changing its attitude, we further tested the UAV by approaching it. From the zoomed-in regions of the trajectories (Fig. 6.3.2 (g)), it could be observed that the UAV had tried to maintain a safety distance, D_{safe}. This shows that the robot was able to justify its relative position not only in terms of 2D image plane, but also in 3D space.

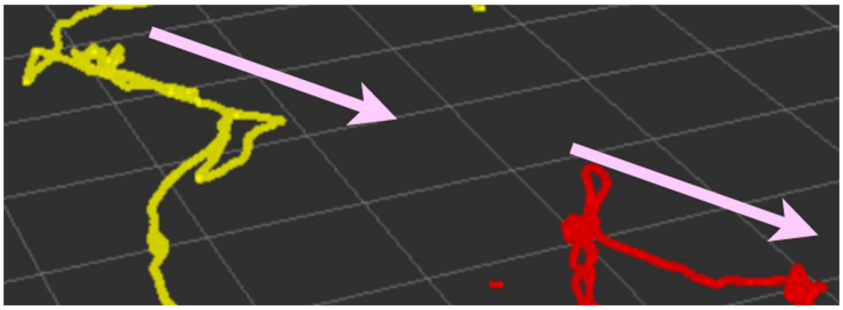


Fig. 6.3.2 (g) the period when UAV tried to move backwards and maintain a safety distance.

Another objective of the experiment is to observe the performance of the Kalman filter. As mentioned, the Kalman filter plays a crucial role when the object is occluded or not detected by YOLO-Tiny. During the experiment, we also let crew members walk between the object and the quadrotor and it showed no sign in losing the object-to-be-tracked, which could be observed in the trajectories. Yet, to further ensure that Kalman filter worked appropriately during the flight test, among the video frames recorded from the FoV of the quadrotor, we have selected several frames where the object was either visually blocked or not detected by YOLO-Tiny (Fig. 6.3.2 (h)).

Fig. 6.3.2 (h) the red boxes were estimated by Kalman filter.

In particular, the frames near the centre sub-image of Fig. 6.3.2 (h) are presented in a series of images in Fig. 6.3.2 (i) below.

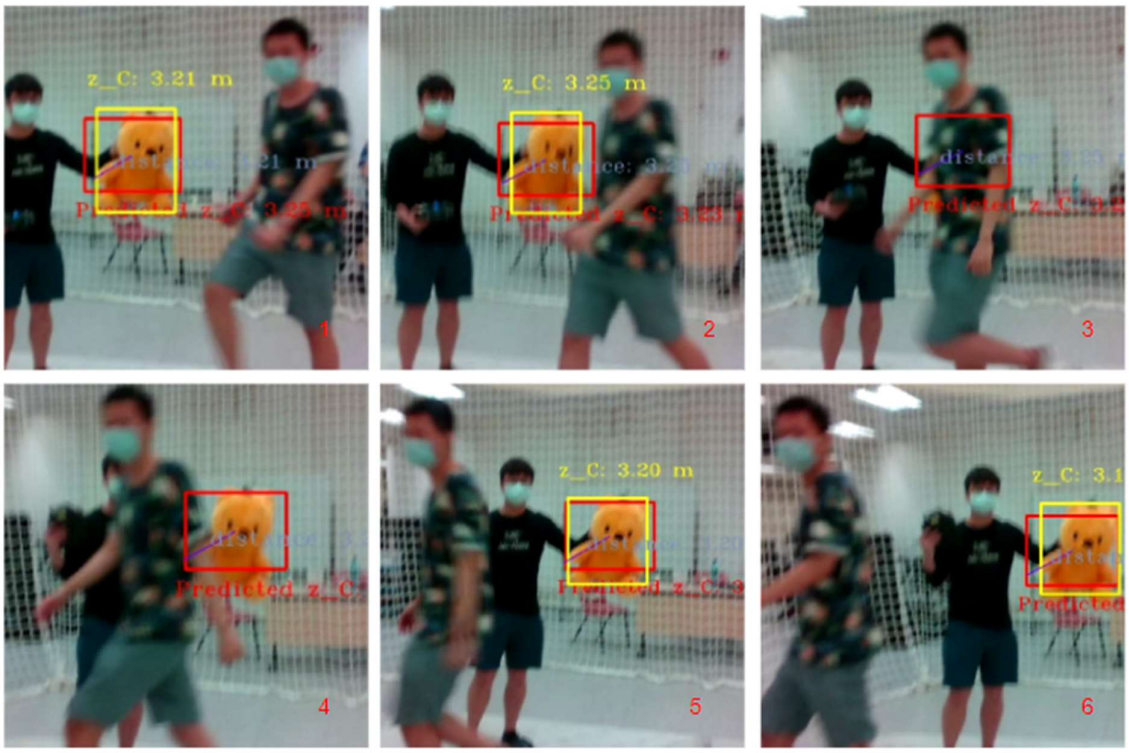


Fig. 6.3.2 (i) the series of frames when object was occupied by a foreign object

From above, it can be seen that the Kalman filter, for the majority of the time, had successfully, or approximately, estimated the object's state, despite the camera's continuous movement. The experiments have been conducted several times, and such satisfactory results imply, to some extent, the robustness of the designed system. Yet, from our observation, when the object produces aggressive or nonlinear movement, such as a sudden change in direction, the Kalman filter cannot work in such efficiency. This indicates that an Extended Kalman filter or Unscented Kalman filter could suit the system better when the object is expected to move nonlinearly. In addition to that, in this paper, we did not take the ego-motion of the camera into account when setting the Kalman filter state vector. With the predefined relatively slow and constant speed of the UAV's motion, the system performed well; yet if the robot moves in a much aggressive fashion, our system might not be as reliable. Therefore, there is a room for further improvement, which will be discussed in Section 7.3.

We strongly recommend readers to review the recorded videos of our experiment, whose links are attached in the Section 10 (Appendix I).

7. Future recommendations

7.1 Merging localization module in both proposed system

In our presented methodologies and conducted tests, we did not consider the independent localization problem as our discussing scope. In our specific case, we employed VICON as an external localization device to help the UAV determine its location at every time epoch. However, VICON environment is only limited to a particular space in the indoor environment. Thus, if our UAV object searching or object tracking algorithms are expected to extend to real outdoor or indoor applications, we could further incorporate GNSS (Global Navigation Satellite System) into our UAV system. GNSS, when available, can provide outdoor localization information; yet GNSS exists signal blockage and signal reflection issues, especially in the crowded urban environment, complex terrain scenario or indoor environment, in which the localization accuracy would be significantly degraded. Therefore, in order to tackle the situation where GNSS is not available, a fully autonomous UAV system based on independent localization is deemed as a further improvement direction for our UAV system.

Using a single IMU (Inertial Measurement Unit) to do localization will possess measurement drift problem and therefore, the measurement will gradually deviate from the authentic information. To cope with this problem, some researchers proposed to employ sensor fusion methods to fuse information from various sensors (e.g., visual camera, radar or ultrasonic sensor) to perform independent localization. Chen et al. (2020) has already proposed a stereo visual-inertial pose estimation method to realize a robust autonomous UAV system. Bloesch et al. (2015) successfully leveraged monocular visual-inertial odometry based on Extended Kalman Filter to accurately estimate the position information of UAV. These researchers can give us future direction for further developing independent UAV localization system.

7.2 Path Planning and further Optimization of Trajectory

In this project, our path planning was designed to avoid column-wise obstacles. Meanwhile, our current trajectory generation is only based on local waypoints. Yet, with the foundation of the work presented here, we considered that it is possible for the interested developers to work in general path planning problems. Such performance could be done if we conduct collision check continuously when avoiding obstacles so that an obstacle with a complex structure could also be averted. We could also possibly do trajectory optimization more frequently by incorporating both global waypoints and the newly encountered local waypoints in our

algorithm. Nevertheless, to achieve so, it is also asserted that the UAV should not have aggressive flight as motion blur might occurred, and the image-process-technique-based planner might behave adversely due to the motion blur and further lead to bad navigation.

In Section 4.3.4, where we employed minimum jerk method to optimize trajectory, we only considered equality constraints such as velocity and acceleration. Yet, our algorithm could be more robust if we add inequality constraints such as corridor check. Corridor check is a way to regulate the shape of a trajectory by adding position constraints at every sampling point on the trajectory such that the trajectory points will lie within an area. By adding the corridor check, our generated trajectory might be closer to the route given by the path planning algorithm. Additionally, as mentioned in Section 6.2.1, the velocities of UAV during simulation were occasionally acutely changed, which may not be energy-efficient and physically applicable for UAV in some circumstances. To resolve this issue, a feasibility check could work. Noting that a feasibility check does not mean simply adding a velocity or acceleration restraint (like a velocity or acceleration corridor). Instead, a feasibility check is essentially correlated with time allocation issue, which is a critical yet intractable issue in trajectory generation. Regarding the time allocation, in our minimum jerk algorithm, we only opted for a "naïve" trapezoid velocity time profile, which is actually not optimal. Richter et al. (2016) proposed a time allocation method by adding time penalization, where the time parameter is considered in the cost function. Thereby, through iteratively refining the segment times, the trajectory quality would be enhanced. In addition, Gao et al. (2018) also proposed a new framework, which decouples the trajectory generation into a spatial trajectory and a mapping function, in order to optimize time allocation. Their research work could provide a foundation for us to ameliorate our trajectory quality.

7.3 Object Tracking -The Kalman Filter Family

In this project, we employed Kalman filter to estimate the object states. Since we pre-set a constant and slow speed for our UAV and did not consider aggressive flight motion, a Kalman filter, which mainly works for linear system dynamics, could give a satisfactory performance. Nevertheless, according to Chowdhary et al. (2014), reality dynamics are essentially non-linear. If we encounter a nonlinear object or demand an aggressive motion of UAV, we cannot guarantee the performance of object tracking which only uses Kalman filter. Here, we deem the Extended Kalman filter (EKF) and Unscented Kalman filter (UKF) as two potentially

useful algorithms to cope with non-linearities in order to further enhance the robustness of our object tracking performance.

For instance, if UAV is expected to perform moderately nonlinear maneuvers such as gentle velocity variations, it will be fairly acceptable to leverage EKF, which will linearize the nonlinear state transition or measurement function around the mean of current state estimate. This method works well for moderate non-linearity, but it is not optimal if the system is highly nonlinear, since the local linearization does not provide a suitable approximation for highly nonlinear system. In contrast, if we require agile UAV motions such as sharp turning and aggressive velocity change, the UKF can be applied to handle this highly nonlinear dynamics. UKF uses a deterministic sampling technique to pick a minimal set of sample points around the mean such that the mean and covariance of these sample points are the same as original probability distribution (e.g., Gaussian distribution). These sample points are then propagated through the nonlinear system model. Subsequently, the mean and covariance of the nonlinearly transformed points are calculated and an empirical Gaussian distribution will be computed, which is then used to calculate the new state estimate. For more information about EKF and UKF, we refer the reader to the paperwork by Wan, et al. (2001).

7.4 Object Tracking whilst Avoiding Obstacles

Our rationale for designing such two systems is motivated by real-time SAR and reconnaissance applications, and in this project, we have designed a separated object searching and object tracking UAV system. At the current stage, the UAV system could not avoid obstacles while tracking objects, because we focus on evaluating how the UAV would act with the motion of the tracking object. Yet, by integrated the developed Edge Detector Planner with object tracking system, it is deemed that the UAV would be able to perform object following motions while avoiding obstacles at the same time. Such combination is considered to allow our object tracking to be employable under situation where there exist foreign objects. This could be a potential future research.

8. Conclusion

In this paper, we have presented a learning-based navigation system for quadrotor UAV to autonomously achieve object searching and object tracking in real time. We proved that the recent developments in the field of computer vision, image processing technology, and UAV robotics can be integrated in a novel UAV system for object searching task and object tracking task.

We have evaluated all modules of our proposed system through extensive experiments in both virtual and real environment. The systematic analysis of experimental results has validated the feasibility and reliability of such system. For object searching, it is capable of accurately identifying the occurrence and location of target objects in an unknown environment with an autonomous and collision-free flight. On the other hand, the object tracking system is capable of consistently tracking and safely following the movement of a single target object without human intervention. Our UAV system demonstrates fairly satisfactory robustness and superiority in low costs at all times.

All the hardware platforms and software configurations for our system have been outlined. The limitations of our works are also discussed with recommendations for future research.

In sum, the objectives of this project are successfully fulfilled with satisfactory achievements. We believe that our effective method configures and eases the future research in this field of study and thus makes a step towards autonomous object searching and object tracking applications in real world scenarios of SAR operations or reconnaissance works.

9. References

A. (n.d.). *AlexeyAB/darknet*. GitHub. Retrieved April 16, 2021, from https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects

Adarsh, P., Rathi, P., & Kumar, M. (2020, March). YOLO v3-Tiny: Object Detection and Recognition using one stage improved model. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*(pp. 687-694). IEEE.

Ahn, M. S., Chae, H., Noh, D., Nam, H., & Hong, D. (2019, June). Analysis and noise modeling of the Intel RealSense D435 for mobile robots. In *2019 16th International Conference on Ubiquitous Robots (UR)* (pp. 707-711). IEEE.

Al-Kaff, A., Gómez-Silva, M. J., Moreno, F. M., de la Escalera, A., & Armingol, J. M. (2019). An appearance-based tracking algorithm for aerial search and rescue purposes. *Sensors*, *19*(3), 652.

Andriluka, M., Schnitzspan, P., Meyer, J., Kohlbrecher, S., Petersen, K., Von Stryk, O., ... & Schiele, B. (2010, October). Vision based victim detection from unmanned aerial vehicles. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 1740-1747). IEEE.

Avidan, S. (2004). Support vector tracking. *IEEE transactions on pattern analysis and machine intelligence*, *26*(8), 1064-1072.

Bejiga, M. B., Zeggada, A., Nouffidj, A., & Melgani, F. (2017). A convolutional neural network approach for assisting avalanche search and rescue operations with UAV imagery. *Remote Sensing*, *9*(2), 100.

Bengio, Y. (2009). *Learning deep architectures for AI*. Now Publishers Inc.

Bernardin, K., & Stiefelhagen, R. (2008). Evaluating multiple object tracking performance: the clear mot metrics. *EURASIP Journal on Image and Video Processing*, *2008*, 1-10.

Bernardin, K., Elbs, A., & Stiefelhagen, R. (2006, May). Multiple object tracking performance metrics and evaluation in a smart room environment. In *Sixth IEEE International Workshop on Visual Surveillance, in conjunction with ECCV*(Vol. 90, p. 91).

Bertalmio, M., Sapiro, G., & Randall, G. (2000). Morphing active contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(7), 733-737.

Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016, September). Simple online and realtime tracking. In *2016 IEEE international conference on image processing (ICIP)* (pp. 3464-3468). IEEE.

Bloesch, M., Omari, S., Hutter, M., & Siegwart, R. (2015, September). Robust visual inertial odometry using a direct EKF-based approach. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 298-304). IEEE.

Bochkovskiy, A. (2020). Darknet: Open Source Neural Networks in Python. Retrieved online from https://github.com/AlexeyAB/darknet

Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.

Breitenstein, M. D., Reichlin, F., Leibe, B., Koller-Meier, E., & Van Gool, L. (2009, September). Robust tracking-by-detection using a detector confidence particle filter. In *2009 IEEE 12th International Conference on Computer Vision* (pp. 1515-1522). IEEE.

Brownlee, J. (2020, August 14). What is the Difference Between Test and Validation Datasets? Machine Learning Mastery. https://machinelearningmastery.com/difference-test-validation-datasets/

Burke, J. L., & Murphy, R. R. (2004, September). Human-robot interaction in USAR technical search: Two heads are better than one. In *RO-MAN 2004. 13th IEEE International Workshop on Robot and Human Interactive Communication (IEEE Catalog No. 04TH8759)* (pp. 307-312). IEEE.

Canny Edge Detection — OpenCV-Python Tutorials 1 documentation. (n.d.). OpenCV-Python Tutorials. Retrieved April 23, 2021, from https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6), 679-698.

Cao, X., Zou, X., Jia, C., Chen, M., & Zeng, Z. (2019). RRT-based path planning for an intelligent litchi-picking manipulator. *Computers and electronics in agriculture*, *156*, 105-118.

Carfagni, M., Furferi, R., Governi, L., Santarelli, C., Servi, M., Uccheddu, F., & Volpe, Y. (2019). Metrological and critical characterization of the Intel D415 stereo depth camera. *Sensors*, *19*(3), 489.

Carrillo, L. R. G., López, A. E. D., Lozano, R., & Pégard, C. (2012). Combining stereo vision and inertial navigation system for a quad-rotor UAV. *Journal of intelligent & robotic systems*, *65*(1), 373-387.

Carrio, A., Sampedro, C., Rodriguez-Ramos, A., & Campoy, P. (2017). A review of deep learning methods and applications for unmanned aerial vehicles. *Journal of Sensors*, *2017*.

Carsten, J., Ferguson, D., & Stentz, A. (2006, October). 3d field d: Improved path planning and replanning in three dimensions. In *2006 IEEE/RSJ international conference on intelligent robots and systems* (pp. 3381-3386). IEEE.

Chan, T. F., & Vese, L. A. (2001). Active contours without edges. *IEEE Transactions on image processing*, *10*(2), 266-277.

Chao, N., Liu, Y. K., Xia, H., Ayodeji, A., & Bai, L. (2018). Grid-based RRT∗ for minimum dose walking path-planning in complex radioactive environments. *Annals of Nuclear Energy*, *115*, 73-82.

Chen, H., & Lu, P. (2020). Computationally Efficient Obstacle Avoidance Trajectory Planner for UAVs Based on Heuristic Angular Search Method. *arXiv preprint arXiv:2003.06136*.

Chen, S. Y. (2011). Kalman filter for robot vision: a survey. *IEEE Transactions on Industrial Electronics*, *59*(11), 4409-4420.

Chen, S., Chen, H., Zhou, W., Wen, C. Y., & Li, B. (2020). End-to-End UAV Simulation for Visual SLAM and Navigation. *arXiv preprint arXiv:2012.00298*.

Chen, S., Wen, C. Y., Zou, Y., & Chen, W. (2020). Stereo visual inertial pose estimation based on feedforward-feedback loops. *arXiv preprint arXiv:2007.02250*.

Choset, H. (2010). Robotic motion planning: Potential functions. *Robotics Institute, Carnegie Mellon University*.

Chowdhary, G., Frazzoli, E., How, J., & Liu, H. (2014). Nonlinear flight control techniques for unmanned aerial vehicles. *Handbook of Unmanned Aerial Vehicles, Springer, Houten*.

Ciaparrone, G., Sánchez, F. L., Tabik, S., Troiano, L., Tagliaferri, R., & Herrera, F. (2020). Deep learning in video multi-object tracking: A survey. *Neurocomputing*, *381*, 61-88.

Dai, D., & Yang, W. (2010). Satellite image classification via two-layer sparse coding with biased image representation. *IEEE Geoscience and Remote Sensing Letters*, *8*(1), 173-176.

Danelljan, M., Hager, G., Shahbaz Khan, F., & Felsberg, M. (2015). Convolutional features for correlation filter based visual tracking. In *Proceedings of the IEEE international conference on computer vision workshops* (pp. 58-66).

Dedes, G., & Dempster, A. G. (2005, September). Indoor GPS positioning-challenges and opportunities. In *VTC-2005-Fall. 2005 IEEE 62nd Vehicular Technology Conference, 2005.*(Vol. 1, pp. 412-415). IEEE.

Du, J. (2018, April). Understanding of object detection based on CNN family and YOLO. In *Journal of Physics: Conference Series* (Vol. 1004, No. 1, p. 012029). IOP Publishing.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2009). Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, *32*(9), 1627-1645.

Feng, Y., Tse, K., Chen, S., Wen, C. Y., & Li, B. (2021). Learning-Based Autonomous UAV System for Electrical and Mechanical (E&M) Device Inspection. *Sensors*, *21*(4), 1385.

Ferguson, D., & Stentz, A. (2007). Field D*: An interpolation-based path planner and replanner. In *Robotics research* (pp. 239-253). Springer, Berlin, Heidelberg.

Foo, J. L., Knutzon, J., Kalivarapu, V., Oliver, J., & Winer, E. (2009). Path planning of unmanned aerial vehicles using B-splines and particle swarm optimization. *Journal of aerospace computing, Information, and communication*, *6*(4), 271-290.

Gao, F., Wu, W., Pan, J., Zhou, B., & Shen, S. (2018, October). Optimal time allocation for quadrotor trajectory generation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 4715-4722). IEEE.

Gao, M., Du, Y., Yang, Y., & Zhang, J. (2019). Adaptive anchor box mechanism to improve the accuracy in the object detection system. *Multimedia Tools and Applications*, *78*(19), 27383-27402.

García-Rodríguez, J., Flórez-Revuelta, F., & García-Chamizo, J. M. (2009). Representing Non-Rigid Objects with Neural Networks. In *Encyclopedia of Artificial Intelligence* (pp. 1363-1369). IGI Global.

Gertz, E. M., & Wright, S. J. (2003). Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software (TOMS)*, *29*(1), 58-81.

Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).

Gladyshev, P., & Patel, A. (2004). Finite state machine approach to digital event reconstruction. *Digital Investigation*, *1*(2), 130-149.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1, No. 2). Cambridge: MIT press.

Goodrich, M. A., Morse, B. S., Gerhardt, D., Cooper, J. L., Quigley, M., Adams, J. A., & Humphrey, C. (2008). Supporting wilderness search and rescue using a camera-equipped mini UAV. *Journal of Field Robotics*, *25*(1-2), 89-110.

Grogan, S., Pellerin, R., & Gamache, M. (2018). The use of unmanned aerial vehicles and drones in search and rescue operations–a survey. *Proceedings of the PROLOG*.

Grout, I. (2011). *Digital systems design with FPGAs and CPLDs*. Elsevier.

Gurtner, A., Greer, D. G., Glassock, R., Mejias, L., Walker, R. A., & Boles, W. W. (2009). Investigation of fish-eye lenses for small-UAV aerial photography. *IEEE Transactions on Geoscience and Remote Sensing*, *47*(3), 709-721.

Hart, N. (1968). Raphael; A Formal Basis for the Heuristic Determination of Minimum Cost Path. *IEEE Transactions on Systems Science and Cybernetics SSC4*, *4*(2).

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, *37*(9), 1904-1916.

Heritage, G., & Large, A. (Eds.). (2009). *Laser scanning for the environmental sciences*. John Wiley & Sons.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, *313*(5786), 504-507.

Hu, W., Zhou, X., Li, W., Luo, W., Zhang, X., & Maybank, S. (2012). Active contour-based visual tracking by integrating colors, shapes, and motions. *IEEE Transactions on Image Processing*, *22*(5), 1778-1792.

Huang, C., Wu, B., & Nevatia, R. (2008, October). Robust object tracking by hierarchical association of detection responses. In European Conference on Computer Vision (pp. 788-801). Springer, Berlin, Heidelberg.

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., ... & Murphy, K. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*(pp. 7310-7311).

Huang, K. Q., Ren, W. Q., & Tan, T. N. (2014). A review on image object classification and detection. *Chinese Journal of Computers*, *37*(6), 1225-1240.

Huang, P., Xu, Y., & Liang, B. (2006). Global minimum-jerk trajectory planning of space manipulator. *International Journal of Control, Automation, and Systems*, *4*(4), 405-413.

Huang, R., Pedoeem, J., & Chen, C. (2018, December). YOLO-LITE: a real-time object detection algorithm optimized for non-GPU computers. In *2018 IEEE International Conference on Big Data (Big Data)* (pp. 2503-2510). IEEE.

Hui, J. (2020a, February 7). mAP (mean Average Precision) for Object Detection - Jonathan Hui. Medium. https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

Hulens, D., & Goedemé, T. (2017, May). Autonomous flying cameraman with embedded person detection and tracking while applying cinematographic rules. In *2017 14th Conference on Computer and Robot Vision (CRV)* (pp. 56-63). IEEE.

Humpherys, J., Redd, P., & West, J. (2012). A fresh look at the Kalman filter. *SIAM review*, *54*(4), 801-823.

Jiang, Z., Zhao, L., Li, S., & Jia, Y. (2020). Real-time object detection method based on improved YOLOv4-tiny. arXiv preprint arXiv:2011.04244.

Johnson, D. B. (1973). A note on Dijkstra's shortest path algorithm. *Journal of the ACM (JACM)*, *20*(3), 385-388.

Karpathy, A., Li, F. F., & Johnson, J. (2017). Cs231n convolutional neural networks for visual recognition (2016). URL http://cs231n. github. io, 50.

Kong, T., Sun, F., Liu, H., Jiang, Y., Li, L., & Shi, J. (2020). Foveabox: Beyound anchor-based object detection. *IEEE Transactions on Image Processing*, *29*, 7389-7398.

Koyuncu, H., & Yang, S. H. (2010). A survey of indoor positioning and object locating systems. *IJCSNS International Journal of Computer Science and Network Security*, *10*(5), 121-128.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84-90.

Kuffner, J. J., & LaValle, S. M. (2000, April). RRT-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)* (Vol. 2, pp. 995-1001). IEEE.

Kushleyev, A., Mellinger, D., Powers, C., & Kumar, V. (2013). Towards a swarm of agile micro quadrotors. *Autonomous Robots*, *35*(4), 287-300.

Kyrkou, C., Plastiras, G., Theocharides, T., Venieris, S. I., & Bouganis, C. S. (2018, March). DroNet: Efficient convolutional neural network detector for real-time UAV applications. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 967-972). IEEE.

Lee, B. Y., Liew, L. H., Cheah, W. S., & Wang, Y. C. (2014, February). Occlusion handling in videos object tracking: A survey. In *IOP conference series: earth and environmental science* (Vol. 18, No. 1, p. 012020). IOP Publishing.

Li, E. Y. (2019). Dive Really Deep into YOLO v3: A Beginner's Guide. *Medium. com*, *31*.

Li, X., Hu, W., Shen, C., Zhang, Z., Dick, A., & Hengel, A. V. D. (2013). A survey of appearance models in visual object tracking. *ACM transactions on Intelligent Systems and Technology (TIST)*, *4*(4), 1-48.

Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision* (pp. 2980-2988).

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In *European conference on computer vision* (pp. 21-37). Springer, Cham.

Liu, Y., Yeoh, J. K., & Chua, D. K. (2020). Deep Learning–Based Enhancement of Motion Blurred UAV Concrete Crack Images. *Journal of Computing in Civil Engineering*, *34*(5), 04020028.

Lu, H. C., Li, P. X., & Wang, D. (2018). Visual object tracking: a survey. *Pattern Recognition and Artificial Intelligence*, *31*(1), 61-76.

Lu, Y., Xue, Z., Xia, G. S., & Zhang, L. (2018). A survey on vision-based UAV navigation. *Geo-spatial information science*, *21*(1), 21-32.

Lumelsky, V. J., & Stepanov, A. A. (1987). Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, *2*(1), 403-430.

Lygouras, E., Santavas, N., Taitzoglou, A., Tarchanidis, K., Mitropoulos, A., & Gasteratos, A. (2019). Unsupervised human detection with an embedded vision system on a fully autonomous uav for search and rescue operations. *Sensors*, *19*(16), 3542.

Mao, Q. C., Sun, H. M., Liu, Y. B., & Jia, R. S. (2019). Mini-YOLOv3: real-time object detector for embedded applications. *IEEE Access*, *7*, 133529-133538.

Masehian, E., & Amin-Naseri, M. R. (2004). A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning. *Journal of Robotic Systems*, *21*(6), 275-300.

McGuire, K. N., de Croon, G. C. H. E., & Tuyls, K. (2019). A comparative study of bug algorithms for robot navigation. *Robotics and Autonomous Systems*, *121*, 103261.

Mellinger, D., & Kumar, V. (2011, May). Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE international conference on robotics and automation* (pp. 2520-2525). IEEE.

Merriaux, P., Dupuis, Y., Boutteau, R., Vasseur, P., & Savatier, X. (2017). A study of vicon system positioning performance. *Sensors*, *17*(7), 1591.

Miller, B., Stepanyan, K., Miller, A., & Andreev, M. (2011, December). 3D path planning in a threat environment. In *2011 50th IEEE Conference on Decision and Control and European Control Conference* (pp. 6864-6869). IEEE.

Mittal, M., Mohan, R., Burgard, W., & Valada, A. (2019). Vision-based autonomous UAV navigation and landing for urban search and rescue. *arXiv preprint arXiv:1906.01304*.

Naveenkumar, M., Sriharsha, K. V., & Vadivel, A. (2018). Moving Object Detection and Tracking Based on the Contour Extraction and Centroid Representation.

In *Encyclopedia of Information Science and Technology, Fourth Edition* (pp. 212-219). IGI Global.

Nikolaidis, N., Krinidis, M., Loutas, E., Stamou, G., & Pitas, I. (2009). Motion tracking in video. In *The Essential Guide to Video Processing* (pp. 175-230). Academic Press.

Ning, G., Zhang, Z., Huang, C., Ren, X., Wang, H., Cai, C., & He, Z. (2017, May). Spatially supervised recurrent convolutional neural networks for visual object tracking. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 1-4). IEEE.

Nixon, M., & Aguado, A. (2019). *Feature extraction and image processing for computer vision*. Academic press.

Oktay, T., Celik, H., & Turkmen, I. (2018). Maximizing autonomous performance of fixed-wing unmanned aerial vehicle to reduce motion blur in taken images. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, *232*(7), 857-868.

OpenCV: Operations on arrays. (n.d.). Open Source Computer Vision. Retrieved April 23, 2021, from https://docs.opencv.org/3.4/d2/de8/group__core__array.html#gaed7df59a3539b4cc0fe5c9c8d7586190

Pehlivanoglu, Y. V., Baysal, O., & Hacioglu, A. (2007). Path planning for autonomous UAV via vibrational genetic algorithm. *Aircraft Engineering and Aerospace Technology*.

Pulford, G. W. (2005). Taxonomy of multiple target tracking methods. *IEE Proceedings-Radar, Sonar and Navigation*, *152*(5), 291-304.

Punn, N. S., Sonbhadra, S. K., & Agarwal, S. (2020). Monitoring COVID-19 social distancing with person detection and tracking via fine-tuned YOLO v3 and Deepsort techniques. *arXiv preprint arXiv:2005.01385*.

Radmanesh, M., Kumar, M., Guentert, P. H., & Sarim, M. (2018). Overview of path-planning and obstacle avoidance algorithms for UAVs: a comparative study. *Unmanned systems*, *6*(02), 95-118.

Ramo, K. (2019). *Hands-On Java Deep Learning for Computer Vision: Implement machine learning and neural network methodologies to perform computer vision-related tasks*. Packt Publishing Ltd.

Rasmussen, C., & Hager, G. D. (2001). Probabilistic data association methods for tracking complex visual objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *23*(6), 560-576.

Rathinam, S., Almeida, P., Kim, Z., Jackson, S., Tinka, A., Grossman, W., & Sengupta, R. (2007, July). Autonomous searching and tracking of a river using an UAV. In *2007 American control conference* (pp. 359-364). IEEE.

Redmon, J. (2016). Darknet: Open Source Neural Networks in C. Retrieved online from http://pjreddie.com/ darknet/

Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7263-7271).

Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. arXiv preprint arXiv:1506.01497.

Ribeiro, M. I. (2004). Kalman and extended kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics*, *43*, 46.

Richter, C., Bry, A., & Roy, N. (2016). Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics research* (pp. 649-666). Springer, Cham.

Ripley, B. D. (2007). Pattern recognition and neural networks. Cambridge university press.

Rudol, P., & Doherty, P. (2008, March). Human body detection and geolocalization for UAV search and rescue missions using color and thermal imagery. In *2008 IEEE aerospace conference* (pp. 1-8). Ieee.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*(3), 211-252.

Ryan, A., & Hedrick, J. K. (2005, December). A mode-switching path planner for UAV-assisted search and rescue. In *Proceedings of the 44th IEEE conference on decision and control* (pp. 1471-1476). IEEE.

Ryll, M., Ware, J., Carter, J., & Roy, N. (2019, May). Efficient trajectory planning for high speed flight in unknown environments. In *2019 International Conference on Robotics and Automation (ICRA)* (pp. 732-738). IEEE.

Schlogl, T., Beleznai, C., Winter, M., & Bischof, H. (2004, August). Performance evaluation metrics for motion detection and tracking. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* (Vol. 4, pp. 519-522). IEEE.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, *61*, 85-117.

Semsch, E., Jakob, M., Pavlicek, D., & Pechoucek, M. (2009, September). Autonomous UAV surveillance in complex urban environments. In *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology* (Vol. 2, pp. 82-85). IEEE.

Shafiee, M. J., Chywl, B., Li, F., & Wong, A. (2017). Fast YOLO: A fast you only look once system for real-time embedded object detection in video. *arXiv preprint arXiv:1709.05943*.

Shi, J. (1994, June). Good features to track. In *1994 Proceedings of IEEE conference on computer vision and pattern recognition* (pp. 593-600). IEEE.

Shomin, M., & Hollis, R. (2014, September). Fast, dynamic trajectory planning for a dynamically stable mobile robot. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 3636-3641). IEEE.

Siam, M., & ElHelw, M. (2012, October). Robust autonomous visual detection and tracking of moving targets in UAV imagery. In *2012 IEEE 11th International Conference on Signal Processing* (Vol. 2, pp. 1060-1066). IEEE.

Škrinjar, J. P., Škorput, P., & Furdić, M. (2018, June). Application of unmanned aerial vehicles in logistic processes. In *International Conference "New Technologies, Development and Applications"* (pp. 359-366). Springer, Cham.

Smeulders, A. W., Chu, D. M., Cucchiara, R., Calderara, S., Dehghan, A., & Shah, M. (2013). Visual tracking: An experimental survey. *IEEE transactions on pattern analysis and machine intelligence*, *36*(7), 1442-1468.

Stentz, A. (1997). Optimal and efficient path planning for partially known environments. In *Intelligent unmanned ground vehicles* (pp. 203-220). Springer, Boston, MA.

Streit, R. L., & Luginbuhl, T. E. (1994, July). Maximum likelihood method for probabilistic multihypothesis tracking. In *Signal and Data Processing of Small Targets 1994* (Vol. 2235, pp. 394-405). International Society for Optics and Photonics.

Sun, J., Li, B., Jiang, Y., & Wen, C. Y. (2016). A camera-based target detection and positioning UAV system for search and rescue (SAR) purposes. *Sensors*, *16*(11), 1778.

Szegedy, C., Toshev, A., & Erhan, D. (2013). Deep neural networks for object detection.

Szeliski, R. 2010. Computer Vision: Algorithms and Applications. London: Springer Science & Business Media.

Tao, H., Sawhney, H. S., & Kumar, R. (2002). Object tracking with bayesian estimation of dynamic layer representations. *IEEE transactions on pattern analysis and machine intelligence*, *24*(1), 75-89.

Tayara, H., & Chong, K. T. (2018). Object detection in very high-resolution aerial images using one-stage densely connected feature pyramid network. Sensors, 18(10), 3341.

Thacker, N. A., & Lacey, A. (1998). Tutorial: The kalman filter. *Imaging Science and Biomedical Engineering Division, Medical School, University of Manchester*, 61.

Tian, S., Liu, X., Liu, M., Li, S., & Yin, B. (2020). Siamese tracking network with informative enhanced loss. *IEEE Transactions on Multimedia*.

Tijtgat, N., Van Ranst, W., Goedeme, T., Volckaert, B., & De Turck, F. (2017). Embedded real-time object detection for a UAV warning system. In *Proceedings of the IEEE International Conference on Computer Vision Workshops* (pp. 2110-2118).

Tomic, T., Schmid, K., Lutz, P., Domel, A., Kassecker, M., Mair, E., ... & Burschka, D. (2012). Toward a fully autonomous UAV: Research platform for indoor and outdoor urban search and rescue. *IEEE robotics & automation magazine*, *19*(3), 46-56.

Uijlings, J. R., Van De Sande, K. E., Gevers, T., & Smeulders, A. W. (2013). Selective search for object recognition. International journal of computer vision, 104(2), 154-171.

Vckay, E., Aneja, M., & Deodhare, D. (2017). Solving a Path Planning Problem in a Partially Known Environment using a Swarm Algorithm. *arXiv preprint arXiv:1705.03176*.

Wallace, L., Lucieer, A., Watson, C., & Turner, D. (2012). Development of a UAV-LiDAR system with application to forest inventory. *Remote sensing*, *4*(6), 1519-1543.

Wan, E. A., & Van Der Merwe, R. (2000, October). The unscented Kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)* (pp. 153-158). IEEE.

Wan, E. A., Van Der Merwe, R., & Haykin, S. (2001). The unscented Kalman filter. *Kalman filtering and neural networks*, *5*(2007), 221-280.

Wang, N., & Yeung, D. Y. (2013). Learning a deep compact image representation for visual tracking. *Advances in neural information processing systems*.

Welch, G. F. (2009). History: The use of the kalman filter for human motion tracking in virtual reality. *Presence: Teleoperators and Virtual Environments*, *18*(1), 72-91.

Welch, G., & Bishop, G. (1995). An introduction to the Kalman filter.

Weng, X., Wang, J., Held, D., & Kitani, K. (2020). 3d multi-object tracking: A baseline and new evaluation metrics. *arXiv preprint arXiv:1907.03961*.

Westall, P., Carnie, R., O'Shea, P., Hrabar, S., & Walker, R. (2007). Vision-based UAV maritime search and rescue using point target detection. In *Proceedings of AIAC12: 12th Australian Aeronautical Conference* (pp. 1-13). Engineers Australia & the Royal Aeronautical Society.

Wise, R., & Rysdyk, R. (2006, August). UAV coordination for autonomous target tracking. In *AIAA Guidance, Navigation, and Control Conference and Exhibit* (p. 6453).

Wojke, N., Bewley, A., & Paulus, D. (2017, September). Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)* (pp. 3645-3649). IEEE.

Xu, S., Savvaris, A., He, S., Shin, H. S., & Tsourdos, A. (2018, June). Real-time implementation of YOLO+ JPDA for small scale UAV multiple object tracking.

In *2018 international conference on unmanned aircraft systems (ICUAS)* (pp. 1336-1341). IEEE.

Yan, F., Liu, Y. S., & Xiao, J. Z. (2013). Path planning in complex 3D environments using a probabilistic roadmap method. *International Journal of Automation and computing*, *10*(6), 525-533.

Yang, L., Qi, J., Xiao, J., & Yong, X. (2014, June). A literature review of UAV 3D path planning. In *Proceeding of the 11th World Congress on Intelligent Control and Automation* (pp. 2376-2381). IEEE.

Yeong, S. P., King, L. M., & Dol, S. S. (2015). A review on marine search and rescue operations using unmanned aerial vehicles. *International Journal of Marine and Environmental Sciences*, *9*(2), 396-399.

Yilmaz, A., Javed, O., & Shah, M. (2006). Object tracking: A survey. *Acm computing surveys (CSUR)*, *38*(4), 13-es.

Zhao, H., Zhou, Y., Zhang, L., Peng, Y., Hu, X., Peng, H., & Cai, X. (2020). Mixed YOLOv3-LITE: A lightweight real-time object detection method. *Sensors*, *20*(7), 1861.

Zhao, Z. Q., Zheng, P., Xu, S. T., & Wu, X. (2019). Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, *30*(11), 3212-3232.

Zhiqiang, W., & Jun, L. (2017, July). A review of object detection based on convolutional neural network. In *2017 36th Chinese Control Conference (CCC)* (pp. 11104-11109). IEEE.

Zhong, Y., Wang, J., Peng, J., & Zhang, L. (2020). Anchor box optimization for object detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (pp. 1286-1294).

Zou, J. T., Wang, C. Y., & Wang, Y. M. (2016, November). The development of indoor positioning aerial robot based on motion capture system. In *2016 International Conference on Advanced Materials for Science and Engineering (ICAMSE)*(pp. 380-383). IEEE.

10. Appendix I

1. Object Searching GitHub link:
   https://github.com/pattylo/Autonomous-Object-Seaching-UAV-system.git

2. Object Searching YouTube video links:
   a. Edge Detector in Gazebo: https://youtu.be/7BZmHzc9BWQ

   b. Object Searching flight test in VICON: https://youtu.be/ZH__uiO9Vo8

c. Object Searching flight test in Gazebo: https://youtu.be/AzSfZh_GZ4A

3. Object Tracking GitHub link:
   https://github.com/pattylo/Autonomous-Object-Tracking-UAV-System.git

4. Object Tracking YouTube video links:
    a. Object Tracking test: https://youtu.be/3VZ3Bd2cdGw

    b. Object Tracking on UAV under VICON: https://youtu.be/9F_7DGWnF5c

11. Appendix II

Summary of Contribution

Project Code:

AAE035

Project Title:

Vision-base of a Quadrotor UAV Navigation

LO, Li-yu (17086854D)

Group mate(s): YIU Chi Hao (17003167D) and Tang Yu (17086039D)

This project mainly consists of the following:

- Object Searching System development:
    - Deep learning literature review
    - YOLO literature review
    - Ubuntu and ROS familiarization
    - YOLO dataset collection, labelling and training
    - YOLO backbone implementation in software architecture
    - Object pose estimation implementation
    - Path planning algorithm design
    - Conduct 3 experiments on YOLO custom training model
    - Conduct 1 experiment on designed path planning
    - Conduct 1 experiment under VICON environment
    - 5 experiment results analysis
- Object Tracking System development
    - Object Tracking literature review
    - YOLO backbone implementation in software architecture
    - Kalman Filter study and implementation
    - Path Planning design
    - Conduct 1 experiment on real-time tracking performance
    - Conduct 1 experiment on UAV platform under VICON.

From above, it could be observed that this project consists of a considerable of work for students and could not be done solely by a man's work. Therefore, at the beginning of the project, we worked together closely on reviewing related literatures and try to decide on the

main topic and the objectives. To be honest, UAV vision-base navigation system could be discussed from various aspects, such as localization, mapping, or path planning, and therefore, it is hard to set a clear direction for us to work on at the early stage. Yet, with the guidance of the subject outline, we first dealt with the learning-based method navigation solution and conduct several literatures on machine learning, deep learning as well as the presented YOLO algorithm in this project. Meanwhile, we also focused on familiarizing the development environment, which were Linux ubuntu operation system and ROS. During the interim report, we proposed to merge YOLO with a 2D path planning method (Bug algorithm) as our main topic, yet it was considered not suitable for our project and hence we move onto further literature review in path planning. After searching and looking for solution, we decided to propose our own path planning method, which is the Edge Detector Planner. Although it might be considered not robust enough, yet, it is also tested to be efficient in column-wise obstacle. Once we have set out our path planning method, we further considered that instead of only searching for a single object, with the robustness of YOLO-Tiny in terms of speed and accuracy, it could also allow the aerial robot to perform tracking and following tasks.

As listed above, this project request abundant literature review. I have personally study most of the topics, yet my focus had been the YOLO backbone, object tracking, Kalman filter and trajectory optimization. In this particular project, my main task is to act as the team leader, where I coordinated all the tasks and host group meetings. In terms of the implementation of the project, I was personally in charge of the software implementation. Specifically, I work closely and frequently with both members on exchanging ideas and discuss the designed algorithm together. Mr. Yiu Chi Hao had worked with me on object tracking system-related coding tasks, whereas Mr. Tang Yu cooperated with me on path planning algorithms. Us three had designed and conducted all 3 experiments together and I mainly in charge of the result analysis on experiment 4, 5 and 7. For this very report, I was in charge of the section 4 and 5, and experiment 4, 5 and 7 in section 6, as mentioned. Us three worked together on introduction, future recommendations as well as conclusion.

I consider that we work all together in this FYP and have contributed equally at all stages.

Project Code:

AAE035

Project Title:

Vision-base of a Quadrotor UAV Navigation

YIU Chi Hao (17003167D)

Group mate(s): LO, Li-yu (17086854D) and Tang Yu (17086039D)

This project mainly consists of the following:

- Object Searching System development:
    - Deep learning literature review
    - YOLO literature review
    - Ubuntu and ROS familiarization
    - YOLO dataset collection, labelling and training
    - YOLO backbone implementation in software architecture
    - Object pose estimation implementation
    - Path planning algorithm design
    - Conduct 3 experiments on YOLO custom training model
    - Conduct 1 experiment on designed path planning
    - Conduct 1 experiment under VICON environment
    - 5 experiment results analysis
- Object Tracking System development
    - Object Tracking literature review
    - YOLO backbone implementation in software architecture
    - Kalman Filter study and implementation
    - Path Planning design
    - Conduct 1 experiment on real-time tracking performance
    - Conduct 1 experiment on UAV platform under VICON.

At the beginning of the project, we worked together closely on reviewing related background and designed the main objectives. We encountered many difficulties in early stage of project due to the insufficient of knowledge in machine learning area and UAV knowledge. We discussed the detail of project in many meetings with professors and teaching assistants. We worked on deep-learning-based YOLO framework and setup of Linux ubuntu operation system and ROS. During the interim report, we proposed to merge YOLO with a 2D path planning

method (Bug algorithm) as our main topic of navigation, yet it was considered not suitable after the interim period. After searching and looking for solution, we decided to propose our own path planning method, which is the Edge Detector Planner to avoid column-wise obstacle. Finally, we decided our project scope as object searching and object tracking, and further worked on it.

In this project, my major mission was to conduct all detailed review on both deep learning based object detection (from CNN, R-CNN to YOLO), object searching and object tracking topics. With a relatively weak foundation on computer science knowledge, I considered the work relatively difficult. However, with consistently works and the discussion and sharing among group members, we still manage to have a certain extent of understanding on the topic and achieve the objectives. In addition to the considerable amount of literature reviews, I also worked closely with group member Tang Yu on the YOLO model training, as I prepared the dataset and Mr. Tang conduct the training process. Furthermore, I had been assisting Mr. LO, Li-yu on the coding implementation of object searching and tracking. For report writing, I was fully in charge of all section of final report, literature review, system overview, methodologies as well as the result analysis in experiment 1, 2, 3 and 6. We also all work together in introduction, future recommendation and conclusion.

I consider that we work all together in this FYP and have contributed equally in all stages. Everyone spent hard efforts in all stages of work.

Project Code:

AAE035

Project Title:

Vision-base of a Quadrotor UAV Navigation

Tang Yu (17086039D)

Group mate(s): LO, Li-yu (17086854D) and YIU Chi Hao (17003167D)

This project mainly consists of the following:

- Object Searching System development:
    - Deep learning literature review
    - YOLO literature review
    - Ubuntu and ROS familiarization
    - YOLO dataset collection, labelling and training
    - YOLO backbone implementation in software architecture
    - Object pose estimation implementation
    - Path planning algorithm design
    - Conduct 3 experiments on YOLO custom training model
    - Conduct 1 experiment on designed path planning
    - Conduct 1 experiment under VICON environment
    - 5 experiment results analysis
- Object Tracking System development
    - Object Tracking literature review
    - YOLO backbone implementation in software architecture
    - Kalman Filter study and implementation
    - Path Planning design
    - Conduct 1 experiment on real-time tracking performance
    - Conduct 1 experiment on UAV platform under VICON.

Our project contained a large amount of work and we basically learned every piece of knowledge from zero foundation. Specifically, we only acquired very little knowledge regarding robotics and computer science at the very beginning of this project. Thus, it was a tough moment for us initially to learn very basic things such as Ubuntu operation system, ROS functions, OpenCV libraries, etc. We also enriched our coding skills based on our limited experience in C++. This difficult project is a big challenge yet an impetus for us to enhance

our technical skills and problem-solving ability at the same time. Our team worked very closely to learn new skills together and cope with every simple problem cooperatively. In particular, I concentrated on learning online courses regarding machine learning and deep learning to have fundamental ideas regarding artificial intelligence before doing literature study. Meanwhile, based on my limited knowledge concerning path planning and trajectory generation, I assiduously made up my knowledge gap in this area by studying a lot of tutorials and literatures to comprehensively enhance my ability in this field, as well as searching for potentially applicable algorithms for our own project. Luckily, our team divided work clearly, cooperated very efficiently, and worked very actively. Thus, we overcame the tough initial stage and gradually adjusted to the intense research work, and hence became more adamant to challenge myself and make some achievements in this project.

After the intermediate stage of our project, we encountered several difficult problems. For example, our originally proposed path planning algorithm cannot feasibly handle our 3-D working scenario. Our trained model was also not satisfactory with a poor real-time detection performance and few class variety. Hence, our group had many discussions and searched a number of technical resources during this period. We eventually developed our own path planning algorithm together based on our extensive literature review. Although our algorithm is neither very mature nor sufficiently robust, it can still satisfy our project scenario requirement. We also significantly refined our model training results after putting extra effort. During this period, I worked frequently with Mr. LO, Li-yu on the algorithm design of the path planning and helped with the coding implementation stage. In addition to this, I also worked with Mr. Yiu Chi Hao regarding the YOLO-Tiny custom model training, in which he prepared the dataset, whereas I was in charge of training the model on Google Colab. Although there is still room for future improvement, this project laid a solid foundation for us in terms of technical capabilities, experience, and confidence. Therefore, I still deem these encountered challenges very common and valuable experiences because only after tackling these problems can we achieve a worthy result.

When it comes to this final report writing, I think three members in our group contributed equally and all significantly to this report completion. Everyone has given numerous constructive ideas in report writing no matter in overall framework and detailed contents. In addition to this, after completing the draft of this report, all of our group members spent a very huge amount of time to do detailed and thorough checking. Each member has exhaustively

checked the report and cooperatively figured out any encountered problems and corrections. Every member really spent a considerable amount of effort both during independent writing stage and cross-check stage. So, our team really worked like a harmonious, enthusiastic, motivated, collaborative, and hard-working group, in which each one would like to share their views, give their assistance, and put their effort. Therefore, to conclude, the achievement of this overall project as well as this final report really comes from everyone's initiative and hard work, and everyone's work is highly appreciated and valued.