Vision based object detection and tracking on quadrotor platform

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

Master of Technology

by
Ritesh Kumar Halder
Roll No.: 16101061

under the guidance of
Dr. Mangal Kothari

Department of Aerospace Engineering
Indian Institute of Technology Kanpur
May, 2018

CERTIFICATE

It is certified that the work contained in this thesis entitled "*Vision based object detection and tracking on quadrotor platform*", by *Ritesh Kumar Halder (Roll No. 16101061)*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Dr. Mangal Kothari)
Department of Aerospace Engineering,
Indian Institute of Technology Kanpur
Kanpur - 208016

May, 2018

Statement of Thesis Preparation

1. Thesis title: Vision based object detection and tracking on quadrotor platform
2. Degree for which the thesis is submitted: Master of Technology
3. Thesis guide was referred to for preparing the thesis.
4. Specifications regarding thesis format have been closely followed.
5. The contents of the thesis have been organized based on the guidelines.
6. The thesis has been prepared without resorting to plagiarism.
7. All sources used have been cited appropriately.
8. The thesis has not been submitted elsewhere for a degree.

Ritesh Kumar Halder
16101061
Department of Aerospace Engineering
May, 2018

Abstract

Unmanned aerial vehicles have gathered attention recently in various domains owing to their robustness and efficiency, compared to conventional aircrafts. Quadrotors specially are very effective for surveillance purposes. This however comes with a bundle of issues, such as determining the position of the subject, high computation power for detection, obstacle avoidance, control related issues in case of occlusion, etc. This thesis proposes a simple approach to detect and track a desired object or person, and use an UAV to follow it autonomously. The approach seeks zero dependency on the moving target to transmit its position in any form. The method chosen for target recognition is vision-based using a common camera mounted on the quadrotor and deep learning techniques were employed for detection. A simple follow control algorithm is then used to enable the vehicle to follow any path which the target acquires autonomously. Erratic behaviour of human movement was considered and smooth control inputs were generated by employing Kalman Filter/Tracking techniques. This also helped it to function appreciably in occluded and crowded environments. The use of a remote server for the detections task further speeds up the process, since quadrotors have low payload carrying capacity.

To evaluate the performance of the proposed approach, simulation studies are performed for 3D cases. These simulations involves a human model, moving in a fixed trajectory for now, and quadrotor model performing the track and follow operation. Different environment scenarios, involving different trajectories for the human, are also tested upon.

Dedicated to my family.

Acknowledgement

Since no one walks alone on the journey of life and no work can be accomplished without the contributions from various people around, I would like to thank those who joined me, walked beside me, and helped me along the way. Now when I stand at this stage of completion of my thesis work, I would like take this opportunity to express my sincere gratitude towards each and every one of them.

I would like to express my sincere gratitude towards my thesis supervisor, Dr. Mangal Kothari, for his constant support and encouragement. I am grateful for his patient guidance and advice in giving a proper direction to my efforts. I thank the Department of Aerospace Engineering, IIT Kanpur, for providing the necessary infrastructure and a congenial environment for research work.

A big thank you to all my department friends and my wing mates for making my stay in the campus a fun and memorable one. Last, but not the least, I thank my parents for always being there for me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Quadrotors with the ability to detect and follow objects, especially humans have been researched and developed actively in recent years. These have a plentiful application in enhancing security and surveillance system, monitoring activities in manufacturing plants, traffic, and can be used in various daily life activity. Although there are already commercially available quadrotors which are being used by professionals for shooting videos of adventure sports. Yet the current systems rely on the use of a mobile phone or some other beacon for correct tracking and positioning of the human, except for the one introduced by DJI with its Phantom 4 drone, which being a commercial product, is closed source.

A human-following robot requires several techniques such as humans target detection, robot control algorithm and obstacles avoidance. Human detection and tracking is a difficult task in general due to abrupt human object motion, object occlusion and object scale change, and changing object appearance due to changes in illumination and viewpoint, non-rigid deformations, intra-class variability in shape and posture, and potential camera movement, non-overlapping field of views between cameras. Furthermore, movement of camera with tilt and jerks accompanied by motion of the quadrotor make it difficult to achieve a fix over the position of a mobile target such as a human as the quad may lose sight of the target. Furthermore, state of the art techniques for detection and tracking use deep learning and require high computation powers and dedicated hardware to function at a reasonable rate.

1.1 Contributions of this Thesis

There are two main contributions of this thesis. The first is to detect the object based on the user's interest, especially a human, and keep tracking the object in the vision frame till desired. The second is to build a quadrotor control algorithm which keeps on following the object by aligning itself in such a way so that the object is never out of the vision range. This could perform well in crowded as well as slightly occluded conditions.

1.2 Organization of this Thesis

The rest of this thesis is organized as follows. Chapter 2 presents the previous work done in this area. Chapter 3 discusses the various background work related to vision algorithms, stereo processing and other techniques. These are required to understand the basis of our vision based system. Chapter 4 gives us the basic ideas about the workings of ROS and Tensorflow modules, on which most of the project is developed. Chapter 5 then discusses the details of the object detection and tracking methods, distance estimation and filtering techniques. Chapter 6 presents the object follow and control algorithm for the quadrotor and their implementation techniques. Chapter 7 then discusses the experimentation setup, simulation tools used, and presents the results that was achieved. Finally Chapter 8 summarises the whole work and also talks about what can be done further.

Chapter 2

Previous Work

In recent years, the problem of human tracking using drones has received a lot of attention. Juhng-Perng Su et al. [1] and Tayyad Naseer et al. [2] use a stereo camera or a depth camera for multi-rotor drone tracking of a human by detecting a targeted object.

Work by Thomas Muller and Markus Muller [3] uses monocular cameras to track a human who havedifferent color against background colors. A highly efficient tracking procedure is presented which relies on well-known color histograms but uses them in a novel manner. This procedure bases on the calculation of a color weighting vector representing the significances of object colors like a kind of an objects color finger print.

Work by Ashraf Qadir et al. [4] focuses on an unmanned miniature plane tracking an object by detecting the image similar to an image called template in two-dimensional images captured by a monocular camera. The algorithm uses zero mean normalized cross correlation to detect and locate an object in the image. Detection and tracking is autonomously carried out on the payload computer and the system is able to work in two different methods. The first method starts detecting and tracking using a stored image patch. The second method allows the operator on the ground to select the interest object for the UAV to track.

Another work by Imamura [5] tracks a human while detecting a human without the differences of colors and the movements of a target by making use of Histograms

of Oriented Gradients (HOG) features and linear Support Vector Machine (SVM) for ROI classification. A method was proposed that a multi-rotor drone can track a human by processing the two-dimensional images captured by a monocular camera installed on the multi-rotor drone. Furthermore, it can detect human without the differences of colors and movements of a target by using the Histograms of Oriented gradients (HOG) features and the linear Support Vector Machine (SVM). Then, it was shown that the multi-rotor drone could track a human by the proposed method.

In '[6] Comaniciu et al. introduce a new framework for efficient tracking of nonrigid objects by spatially masking the target with an isotropic kernel. The masking induces spatially-smooth similarity functions suitable for gradient-based optimization, hence,the target localization problem can be formulated using the basin of attraction of the local maxima.

In [7] Weng et al. proposed an adaptive Kalman filtering algorithm to effectively track the moving object in a video frame sequence. In initialization, a moving object selected by the user is segmented and the dominant color is extracted from the segmented target. In tracking step, a motion model is constructed to set the system model of adaptive Kalman filter firstly. Then, the dominant color of the moving object in HSI color space will be used as feature to detect the moving object in the consecutive video frames. The detected result is fed back as the measurement of adaptive Kalman filter and the estimate parameters of adaptive Kalman filter are adjusted by occlusion ratio adaptively.

Liu et al. and Redmon et al. have worked on methods using deep learning [8, 9] to detect a bounding box around an object in an image. Named SSD, the approach discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes.

Achtelik et. al. in [10] work on a system where motion of a quadcopter is stably controlled based on visual feedback and measurements of inertial sensors. Active markers were finely designed to improve visibility under various perspectives as well as robustness towards disturbances in the image-based pose estimation. Moreover, position-and heading controllers for the quadcopter were implemented to show the system's capabilities.

Bartak et. al. [11] utilize a computer-vision approach called tracking-learning-detection (TLD) to track an arbitrary object selected by a user in the videostream going from the front camera of the drone. Information about location of the tracked object is then used to guide the drone using the proportional-integral-derivative (PID) controller. The method was implemented in software FollowMe.

Dang et. al. [12] perform a systematic formulation of a closed-loop control system design for tracking a ground object using ardrone platform. In the PhD thesis of Kalal [13] the authors propose a novel tracking framework (TLD) that explicitly decomposes the long-term tracking task into tracking, learning, and detection, where the tracker follows the object from frame to frame. In a seminal work by Lucas and Kanade [14], the authors present a new image registration technique that uses spatial intensity gradient information to direct the search for the position that yields the best match in stereo image pairs.

Chapter 3

Background of vision based detection

3.1 Traditional vision algorithms

Here we describe some tradional techniques used in computer vision. These techniques formulate some way to represent the image by encodeing various features, like corners, edges, color scheme, textures. [15]

3.1.1 *SIFT(Scale invariant feature transform)*

The SIFT algorithm deals with the problem that certain image features like edges and corners are not scale-invariant. In other words, there are times when a corner looks like a corner, but looks like a completely different item when the image is blown up by a few factors. The SIFT algorithm uses a series of mathematical approximations to learn a representation of the image that is scale-invariant. In effect, it tries to standardize all images (if the image is blown up, SIFT shrinks it; if the image is shrunk, SIFT enlarges it). This corresponds to the idea that if some feature (say a corner) can be detected in an image using some square-window of dimension σ across the pixels, then if the image was scaled to be larger, we would need a larger dimension kσ to capture the same corner (see figure 1). The general

idea is that SIFT standardizes the scale of the image then detects important key features. The existence of these features are subsequently encoded into a vector used to represent the image.

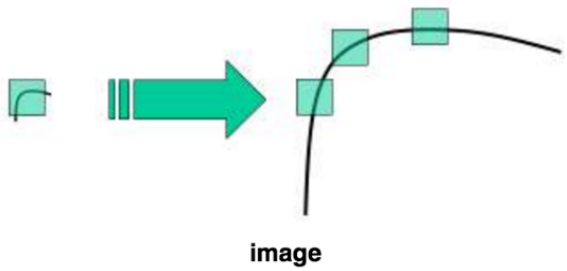

Figure 3.1: Example showing scaling changes detection of a corner

3.1.2 *SURF(Speeded-Up Robust Features)*

The problem with SIFT is that the algorithm itself uses a series of approximations using difference of Gaussians for standardizing the scale. Unfortunately, this approximation scheme is slow. SURF is simply a speeded-up version of SIFT. SURF works by finding a quick and dirty approximation to the difference of Gaussians using a technique called box blur. A box blur is the average value of all the images values in a given rectangle and it can be computed efficiently.

3.1.3 *ORB(Oriented FAST and rotated BRIEF)*

ORB is basically a fusion of FAST keypoint detector and BRIEF descriptor with many modifications to enhance the performance. First it use FAST to find keypoints, then apply Harris corner measure to find top N points among them. It also use pyramid to produce multiscale-features. But one problem is that, FAST doesnt compute the orientation. So what about rotation invariance? Authors came up with following modification.

It computes the intensity weighted centroid of the patch with located corner at center. The direction of the vector from this corner point to centroid gives the

orientation. To improve the rotation invariance, moments are computed with x and y which should be in a circular region of radius r, where r is the size of the patch.

Figure 3.2: ORB features

3.1.4 *HOG(Histogram of oriented gradients)*

The HOG descriptor technique counts occurrences of gradient orientation in localized portions of an image - detection window, or region of interest (ROI). Implementation of the HOG descriptor algorithm is as follows:

Divide the image into small connected regions called cells, and for each cell compute a histogram of gradient directions or edge orientations for the pixels within the cell. Discretize each cell into angular bins according to the gradient orientation. Each cell's pixel contributes weighted gradient to its corresponding angular bin. Groups of adjacent cells are considered as spatial regions called blocks. The grouping of cells into a block is the basis for grouping and normalization of histograms. Normalized group of histograms represents the block histogram. The set of these block histograms represents the descriptor.

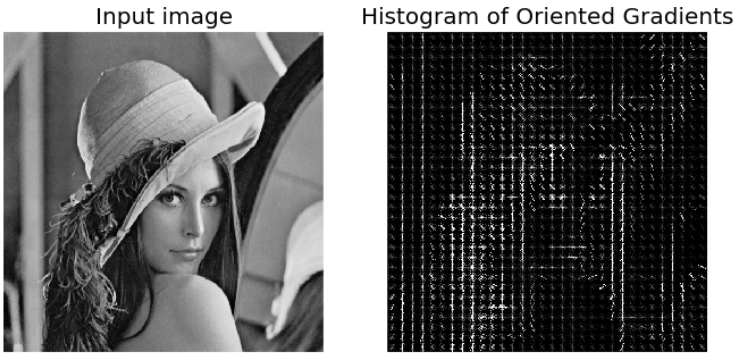

Figure 3.3: HOG features

3.2 Deep Neural Networks

The deep networks we examine here are convolutional neural networks. For those familiar with artificial neural networks, these are simply multi-level neural networks with a few special properties in place (pooling, convolution). The basic idea is that we will take a raw RGB image and perform a series of transformations on the image. On each transformation, we learn a denser representation of the image. We then take this denser representation, apply the transformation and learn an even further denser representation. It turns out by using this procedure, we learn more and more abstract features with which to represent the original image. At the end, we can use these abstract features to predict using some traditional classification method.

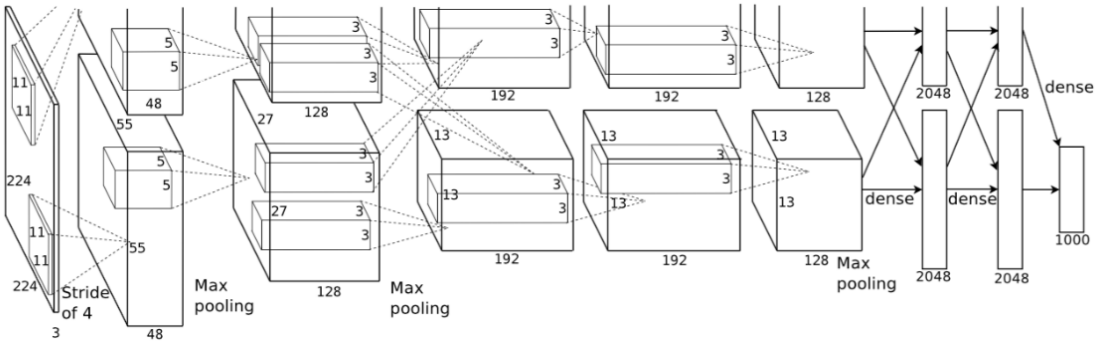

Figure 3.4: CNN training process

3.2.1 *Region-based Convolutional Neural Networks(R-CNN):*

Since we had modeled object detection into a classification problem, success depends on the accuracy of classification. CNNs were too slow and computationally very expensive. It was impossible to run CNNs on so many patches generated by sliding window detector. R-CNN solves this problem by using an object proposal algorithm called *Selective Search* which reduces the number of bounding boxes that are fed to the classifier to close to 2000 region proposals. Selective search uses local cues like texture, intensity, color or a measure of insideness etc to generate all the possible locations of the object. We feed these boxes to our CNN based classifier. Fully connected part of CNN takes a fixed sized input so, we resize(without preserving aspect ratio) all the generated boxes to a fixed size (224 x 224 for VGG) and feed to the CNN part. Hence, there are 3 important parts of R-CNN: [16]

- Run Selective Search to generate probable objects.
- Feed these patches to CNN, followed by SVM to predict the class of each patch.
- Optimize patches by training bounding box regression separately.

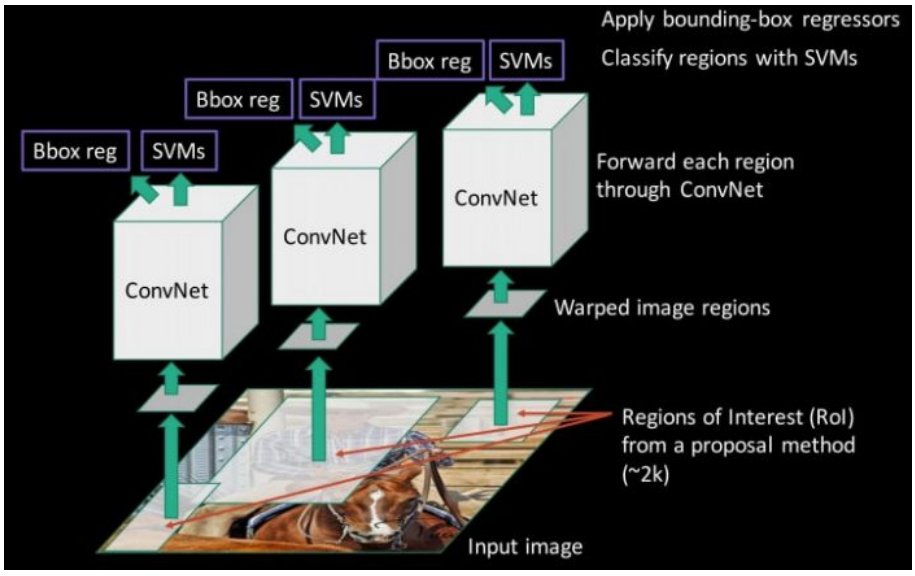

Figure 3.5: R-CNN

3.2.2 *Spatial Pyramid Pooling(SPP-net):*

Still, RCNN was very slow. Because running CNN on 2000 region proposals generated by Selective search takes a lot of time. SPP-Net tried to fix this. With SPP-net, we calculate the CNN representation for entire image only once and can use that to calculate the CNN representation for each patch generated by Selective Search. This can be done by performing a pooling type of operation on JUST that section of the feature maps of last conv layer that corresponds to the region. The rectangular section of conv layer corresponding to a region can be calculated by projecting the region on conv layer by taking into account the downsampling happening in the intermediate layers(simply dividing the coordinates by 16 in case of VGG).

We need to generate the fixed size of input for the fully connected layers of the CNN so, SPP introduces one more trick. It uses spatial pooling after the last convolutional layer as opposed to traditionally used max-pooling. SPP layer divides a region of any arbitrary size into a constant number of bins and max pool is performed on each of the bins. Since the number of bins remains the same, a constant size vector is produced as demonstrated in the figure below. There was one

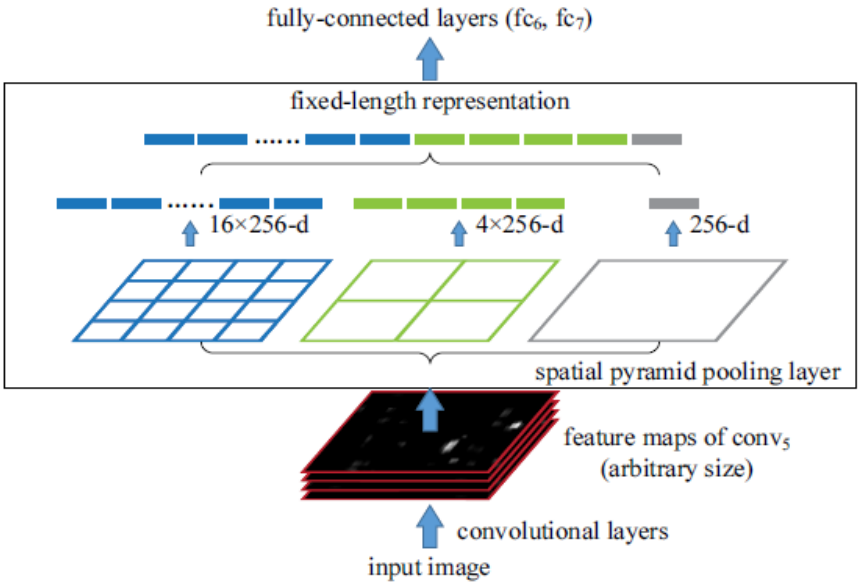

Figure 3.6: SPP-net

big drawback with SPP net, it was not trivial to perform back-propagation through

spatial pooling layer. Hence, the network only fine-tuned the fully connected part of the network. SPP-Net paved the way for more popular Fast RCNN.

3.2.3 *Fast R-CNN*

Fast RCNN uses the ideas from SPP-net and RCNN and fixes the key problem in SPP-net i.e. they made it possible to train end-to-end. To propagate the gradients through spatial pooling, It uses a simple back-propagation calculation which is very similar to max-pooling gradient calculation with the exception that pooling regions overlap and therefore a cell can have gradients pumping in from multiple regions.

One more thing that Fast RCNN did that they added the bounding box regression to the neural network training itself. So, now the network had two heads, classification head, and bounding box regression head. This multitask objective is a salient feature of Fast-rcnn as it no longer requires training of the network independently for classification and localization. These two changes reduce the overall training time and increase the accuracy in comparison to SPP net because of the end to end learning of CNN.

3.2.4 *Faster R-CNN*

Faster RCNN replaces selective search with a very small convolutional network called Region Proposal Network to generate regions of Interests. [17]

To handle the variations in aspect ratio and scale of objects, Faster R-CNN introduces the idea of *anchor boxes*. At each location, the original paper uses 3 kinds of anchor boxes for scale *128*128, 256*256 and 512*512*. Similarly, for aspect ratio, it uses three aspect ratios *1:1, 2:1 and 1:2*. So, in total at each location, we have 9 boxes on which RPN predicts the probability of it being background or foreground. We apply bounding box regression to improve the anchor boxes at each location. So, RPN gives out bounding boxes of various sizes with the corresponding probabilities of each class. The varying sizes of bounding boxes can be passed further by applying Spatial Pooling just like Fast-RCNN. The remaining network is

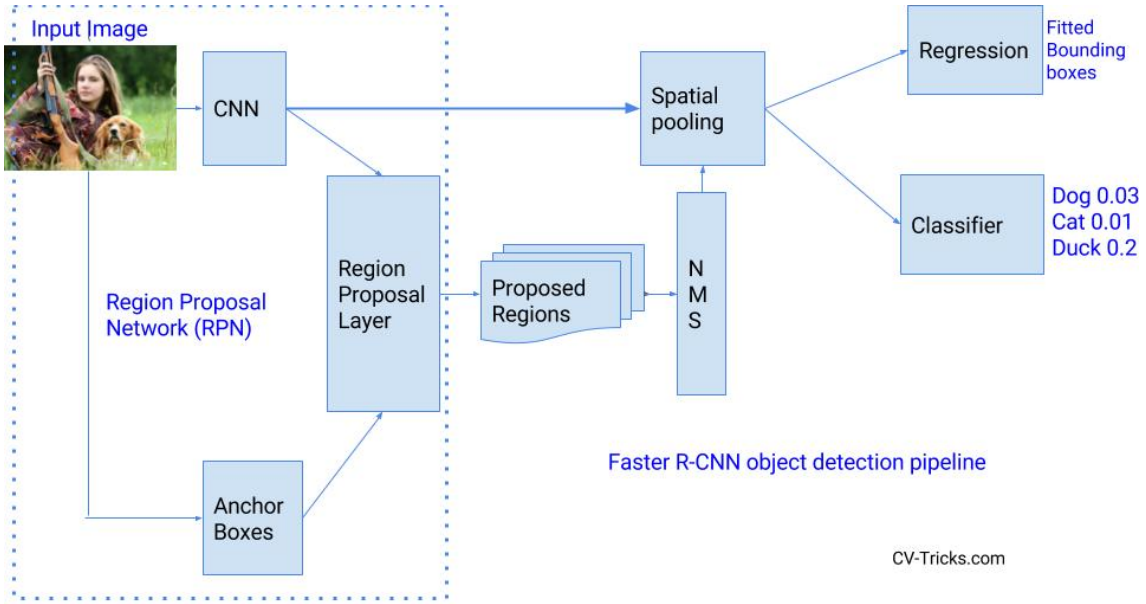

Figure 3.7: Faster R-CNN object detecttion pipeline

similar to Fast-RCNN. Faster-RCNN is 10 times faster than Fast-RCNN with similar accuracy of datasets like VOC-2007. Thats why Faster-RCNN has been one of the most accurate object detection algorithms. Here is a quick comparison between various versions of RCNN.

Category	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image	50s	2s	0.2s
Speed Up	1x	25x	250x

Table 3.1: Comparison between various R-CNNs

3.2.5 *YOLO(You only Look Once):*

YOLO divides each image into a grid of S x S and each grid predicts N bounding boxes and confidence. The confidence reflects the accuracy of the bounding box and whether the bounding box actually contains an object(regardless of class). YOLO also predicts the classification score for each box for every class in training. We can combine both the classes to calculate the probability of each class being present in a predicted box. [16]

So, total SxSxN boxes are predicted. However, most of these boxes have low

confidence scores and if we set a threshold say 30 percent confidence, we can remove most of them as shown in the example below.

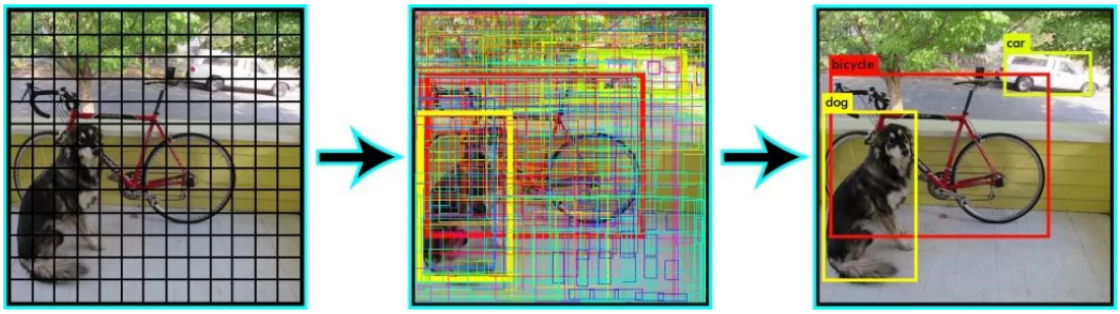

Figure 3.8: YOLO model

At runtime, we have run our image on CNN only once. Hence, YOLO is super fast and can be run real time. Another key difference is that YOLO sees the complete image at once as opposed to looking at only a generated region proposals in the previous methods. So, this contextual information helps in avoiding false positives. However, one limitation for YOLO is that it only predicts 1 type of class in one grid hence, it struggles with very small objects.

3.2.6 *Single Shot Detector(SSD):*

Single Shot Detector achieves a good balance between speed and accuracy. SSD runs a convolutional network on input image only once and calculates a feature map. Now, we run a small 33 sized convolutional kernel on this feature map to predict the bounding boxes and classification probability. SSD also uses anchor boxes at various aspect ratio similar to Faster-RCNN and learns the off-set rather than learning the box. In order to handle the scale, SSD predicts bounding boxes after multiple convolutional layers. Since each convolutional layer operates at a different scale, it is able to detect objects of various scales. [16]

3.3 Relevant literature

3.3.1 ImageNet Classification with Deep Convolutional Neural Networks

This paper discusses a neural network containing over 60 million parameters and 60 million parameters that significantly beat previous state-of-the-art approaches to image recognition in a popular computer vision competition: ISVRC-2012

The architecture of the featured convolutional neural network is given by 5 convolutional layers that are followed by max-pooling layers and 3 full-connected layers with a final 1000-way softmax. Important innovations of this paper include an alternative to the traditional sigmoid activation function and methods for reducing over-fitting. Traditionally, activation functions take the form

$$f(x) = (1 + e^{-x})^{-1} \quad or \quad tanh(x) \tag{3.1}$$

However, the paper advocates the use of *Rectified Linear Units (ReLUs)*, which refers to the activation function

$$f(x) = max(0, x) \tag{3.2}$$

The max function significantly accelerates learning time. To reduce over-fitting, the authors also artificially enlarged the data set by generating image translations and horizontal reflections on each image, while preserving their labels. This increased the training set by a factor of 2048. While the new training examples are highly correlated with the original examples, this data augmentation forces the network to learn multiple representations of the same image, thereby encouraging generalization. CNN described in this paper achieves an error rate of 37.5 as compared to a more traditional approach of using SIFT + FV features, which achieves an error rate of 45.7. [15]

3.4 Comparison Results

Currently, Faster-RCNN is the choice if we are concerned about accuracy numbers. However, if we require better computation(probably running it on Nvidia Jetsons), SSD is a better recommendation. Finally, if accuracy is not too much of a concern but we want to go super fast, YOLO will be the way to go. First of all a visual understanding of speed vs accuracy trade-off can be found in Figure 3.9. [16]

Figure 3.9: YOLO vs SSD

SSD seems to be a good choice as we are able to run it on a video and the accuracy trade-off is very little. This chart compares the performance of SSD, YOLO, and Faster-RCNN on various sized objects. At large sizes, SSD seems to perform similarly to Faster-RCNN. However, when we look at the accuracy numbers for smaller sized objects, the gap widens, as seen in Fig 3.10.

This choice of a right object detection method is crucial and depends on the problem that we are trying to solve and the set-up. Object Detection is the backbone of many practical applications of computer vision such as autonomous cars, security and surveillance, and many other industrial and domestic applications.

In order to solve the problem in this thesis, we have used a SSD based Tensorflow object detection API released some time back.

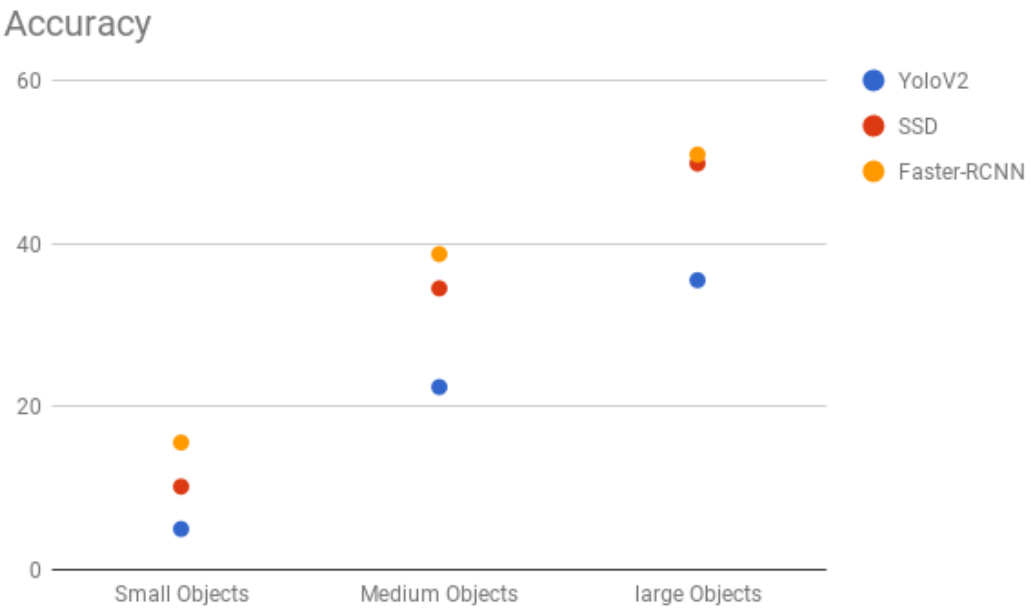

Figure 3.10: YOLO vs SSD(Based on Object size)

Chapter 4

ROS and Tensorflow

4.1 Overview of ROS

ROS (*Robot Operating System*) [18] is a BSD-licensed system for controlling robotic components from a PC. A ROS system is comprised of a number of independent nodes, each of which communicates with the other nodes using a publish/subscribe messaging model. For example, a particular sensors driver might be implemented as a node, which publishes sensor data in a stream of messages. These messages could be consumed by any number of other nodes, including filters, loggers, and also higher-level systems such as guidance, pathfinding, etc.

Why ROS?

Nodes in ROS do not have to be on the same system (multiple computers) or even of the same architecture. We can have a Arduino publishing messages, a laptop subscribing to them, and an Android phone driving motors. This makes ROS really flexible and adaptable to the needs of the user. ROS is also open source, maintained by many people. This is of prime importance in this thesis work.

Lets say we have a camera on our Robot. We want to be able to see the images from the camera, on another laptop. The Camera Node takes care of communication with the camera, Image Processing Node on the robot processes image data, and a Image Display Node displays images on a screen. To start with, all Nodes have registered with the Master. The Master is like a lookup table where all the nodes

go to find where exactly to send messages.

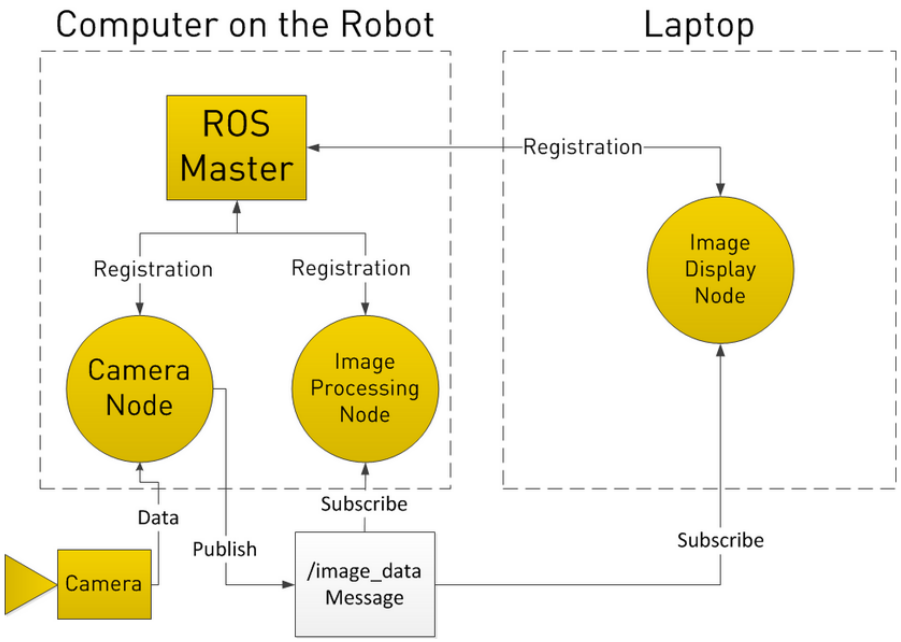

Figure 4.1: ROS working

In registering with the ROS Master, the Camera Node states that it will Publish a Topic called */image-data* (for example). Both of the other Nodes register that they are Subscribed to the Topic */image-data*.

Thus, once the Camera Node receives some data from the Camera, it sends the */image-data* message directly to the other two nodes, on different computer or to the onboard computer.(Through TCP/IP usually)

4.2 Basics of Tensorflow

TensorFlow is a framework created by Google for creating Deep Learning models [19]. Deep Learning is a category of machine learning models that use multi-layer neural networks. The idea of deep learning has been around since 1943 when neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper on how neurons might work and they modeled a simple neural network using electrical circuits.

Basic Computational Graph

Everything in TensorFlow is based on creating a computational graph. Here we have a computational graph as a network of nodes, with each node known as an operation, running some function that can be as simple as addition or subtraction to as complex as some multi variate equation.

In a TensorFlow graph, each node has zero or more inputs and zero or more outputs, and represents the instantiation of an operation. Values that flow along normal edges in the graph (from outputs to inputs) are tensors, arbitrary dimensionality arrays where the underlying element type is specified or inferred at graph-construction time. Special edges, called control dependencies, can also exist in the graph: no data flows along such edges, but they indicate that the source node for the control dependence must finish executing before the destination node for the control dependence starts executing.

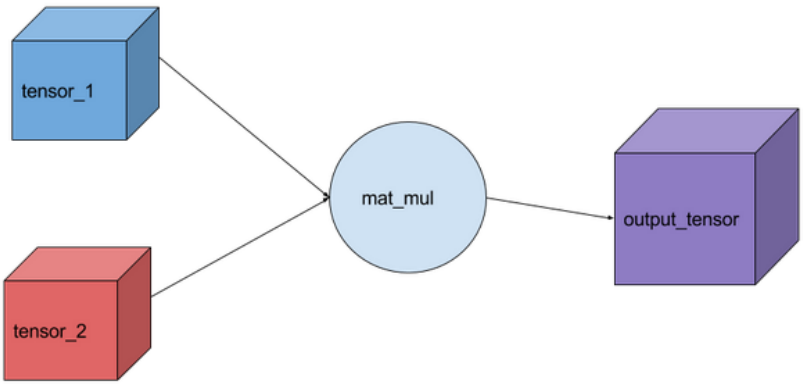

Figure 4.2: Our computation graph

An Operation also referred to as op can return zero or more tensors which can be used later on in the graph.Each operation can be handed a constant, array, matrix or n-dimensional matrix. Another word for an n-dimensional matrix is a tensor, a 2-dimensional tensor is equivalent to a m x m matrix.

Clients programs interact with the TensorFlow system by creating a Session. To create a computation graph, the Session interface supports an Extend method to augment the current graph managed by the session with additional nodes and edges

(the initial graph when a session is created is empty).

Implementation

The main components in a TensorFlow system are the client, which uses the Session interface to communicate with the master, and one or more worker processes, with each worker process responsible for arbitrating access to one or more computational devices(such as CPU cores or GPU cards) and for executing graph nodes on those devices as instructed by the master. We have both local and distributed implementations of the TensorFlow interface. The local implementation is used when the client, the master, and the worker all run on a single machine in the context of a single operating system process (possibly with multiple devices, if for example, the machine has many GPU cards installed). The distributed implementation shares most of the code with the local implementation, but extends it with support for an environment where the client, the master, and the workers can all be in different processes on different machines.

Chapter 5

Object detection and tracking using the Tensorflow API

5.1 Object detection as Image Classification problem

Object detection is modeled as a classification problem. While classification is about predicting label of the object present in an image, detection goes further than that and finds locations of those objects too. In classification, it is assumed that object occupies a significant portion of the image.So it is about finding all the objects present in an image, predicting their labels/classes and assigning a bounding box around those objects. [20]

In image classification, we predict the probabilities of each class, while in object detection, we also predict a bounding box containing the object of that class. So, the output of the network should be:

- Class probabilities (like classification)
- Bounding box coordinates. We denote these by cx(x coordinate of center), cy(y coordinate of center), h(height of object), w(width of object)

Class probabilities should also include one additional label representing background

because a lot of locations in the image do not correspond to any object.

5.1.1 Sliding Window Detector

After the classification network is trained, it can then be used to carry out detection on a new image in a sliding window manner. First, we take a window of a certain size(blue box) and run it over the image at various locations. Then we crop the patches contained in the boxes and resize them to the input size of classification convnet. We then feed these patches into the network to obtain labels of the object. We repeat this process with smaller window size in order to be able to capture objects of smaller size. So the idea is that if there is an object present in an image, we would have a window that properly encompasses the object and produce label corresponding to that object.

5.1.2 Reducing redundant calculations

The following figure-6 shows an image of size 12 x 12 which is initially passed through 3 convolutional layers, each with filter size 3 3(with varying stride and max-pooling). Notice that, after passing through 3 convolutional layers, we are left with a feature map of size 3 x 3 x 64 which has been termed penultimate feature map i.e. feature map just before applying classification layer. We name this because we are going to be referring it repeatedly from here on. On top of this 3 x 3 map, we have applied a convolutional layer with a kernel of size 3 x 3. Three sets of this 3 x 3 filters are used here to obtain 3 class probabilities(for three classes) arranged in 1X1 feature map at the end of the network.

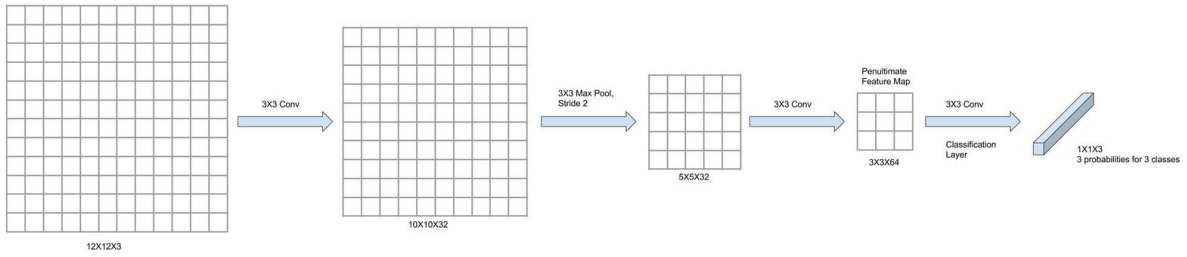

Figure 5.1: Image of 12*12 passed through 3 conv layers

Now, we shall take a slightly bigger image to show a direct mapping between the input image and feature map. Lets increase the image to 14 x 14. We can see that 12 x 12 patch in the top left quadrant(center at 6,6) is producing the 33 patch in penultimate layer colored in blue and finally giving 11 score in final feature map(colored in blue). The second patch of 12 x 12 size from the image located in top right quadrant(shown in red, center at 8,6) will correspondingly produce 1x1 score in final layer(marked in red).

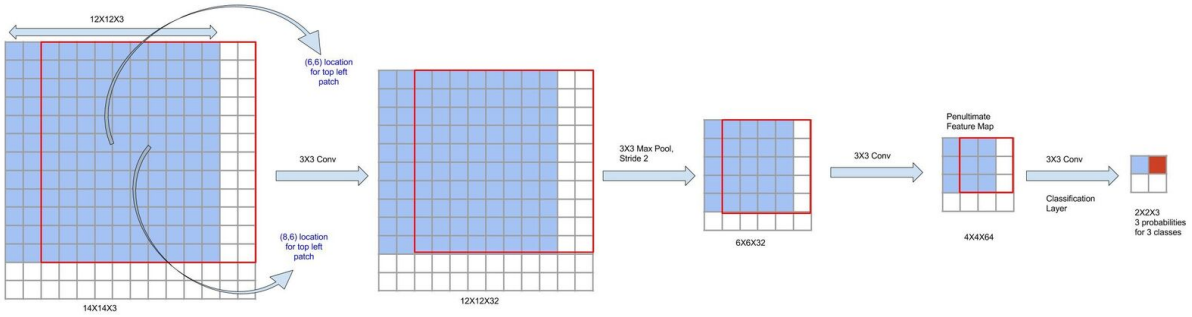

Figure 5.2: Depicting overlap in feature maps for overlapping image regions

To summarize we feed the whole image into the network at one go and obtain feature at the penultimate map. And then we run a sliding window detection with a 3 x 3 kernel convolution on top of this map to obtain class scores for different patches.

5.1.3 Training Methodology for modified network

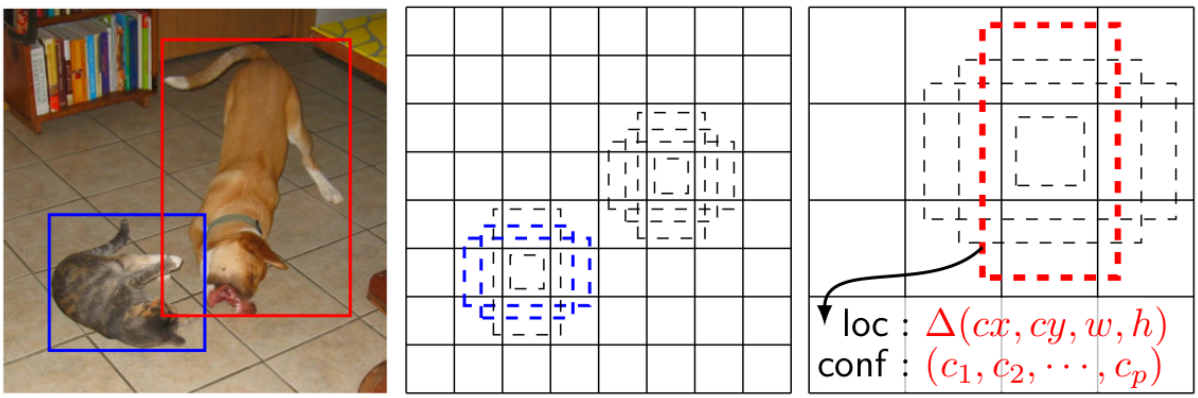

Figure 5.3: Image with GT boxes, 8*8 feature map, 4*4 feature map

SSD [8] only needs an input image and ground truth boxes for each object during training. In a convolutional fashion, we evaluate a small set of default boxes of different aspect ratios at each location in several feature maps with different scales (e.g. 8*8 and 4*4 in the figures) For each default box, we predict both the shape offsets and the confidences for all object categories ($(c_1, c_2, ..., c_p)$). At training time, we first match these default boxes to the ground truth boxes. For example, we have matched two default boxes with the cat and one with the dog, which are treated as positives and the rest as negatives. The model loss is a weighted sum between localization loss (e.g. Smooth L1) and confidence loss (e.g. Softmax).

5.1.4 Working of the SSD Model

The SSD approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes, followed by a non-maximum suppression step to produce the final detections. The early network layers are based on a standard architecture used for high quality image classification (truncated before any classification layers).We then add auxiliary structure to the network to produce detections with the following key features:

- *Multi-scale feature maps for detection:* We add convolutional feature layers to the end of the truncated base network. These layers decrease in size progressively and allow predictions of detections at multiple scales. The convolutional model for predicting detections is different for each feature layer (cf Overfeat and YOLO that operate on a single scale feature map).

- *Convolutional predictors for detection:* Each added feature layer (or optionally an existing feature layer from the base network) can produce a fixed set of detection predictions using a set of convolutional filters. These are indicated on top of the SSD network architecture. The bounding box offset output values are measured relative to a default box position relative to each feature

map location (cf the architecture of YOLO that uses an intermediate fully connected layer instead of a convolutional filter for this step).

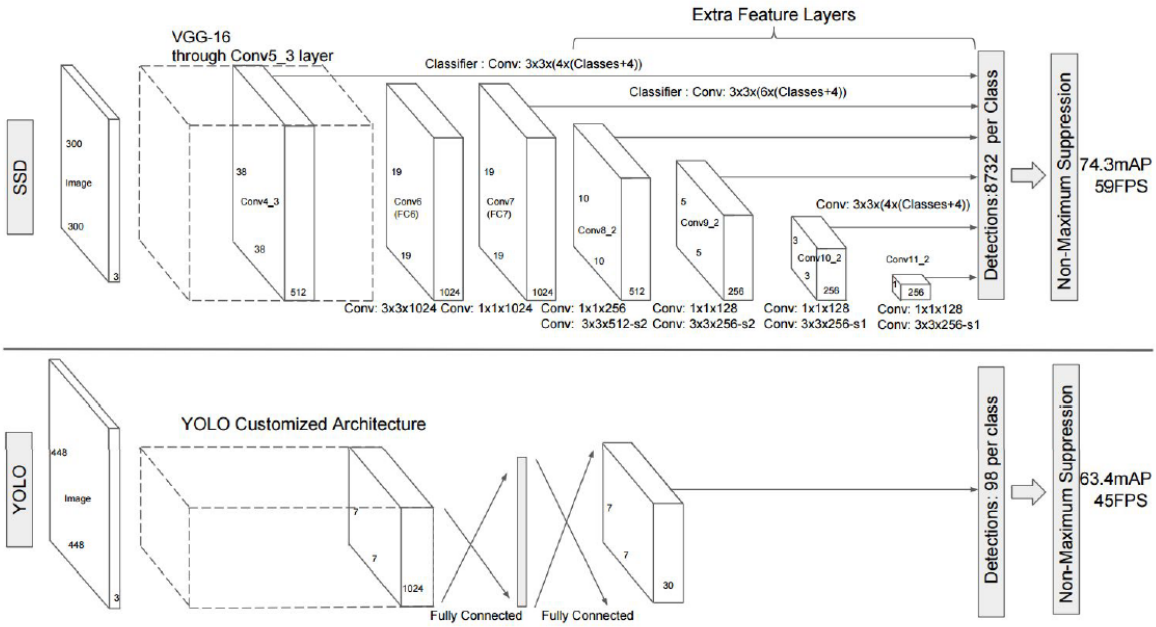

Figure 5.4: A comparison between two single shot detection models: SSD and YOLO

- *Default boxes and aspect ratios"* We associate a set of default bounding boxes with each feature map cell, for multiple feature maps at the top of the network. The default boxes tile the feature map in a convolutional manner, so that the position of each box relative to its corresponding cell is fixed. Allowing different default box shapes in several feature maps let us efficiently discretize the space of possible output box shapes.

5.2 Remote machine object detection node

Deep Learning based approaches trained on the large datasets like Image-Net, such as SSD (Single Shot multi-box Detector) [8], faster R-CNN [17], YOLO (You Only Look Once) [9] etcetera have been consistently performing well for the classification of different objects, including humans. Detection is done using these methods in order to identify humans in the camera image from the drones. A simple comparison

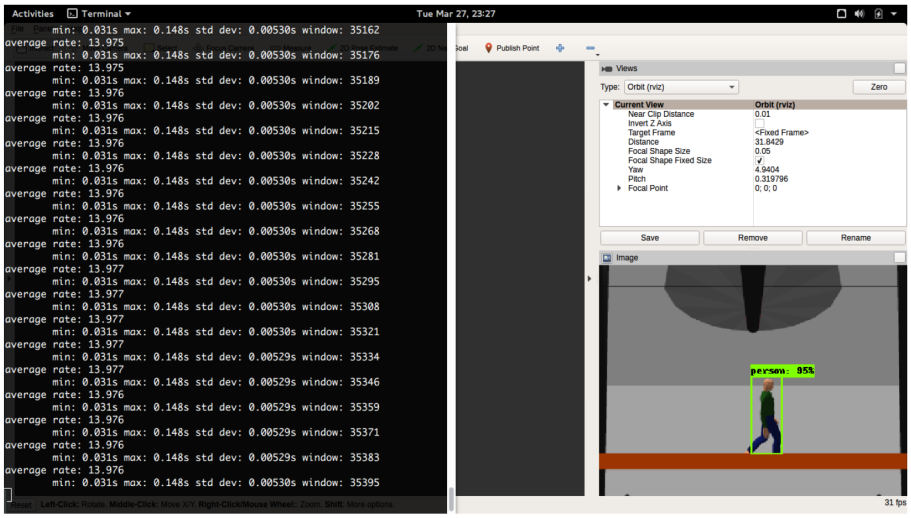

Figure 5.5: Object detection along with rate on remote server

of the speed of the different methods leads to the top contenders, YOLO and SSD. Now, one of the big issues with these approaches, stopping them from real-time application in drones is the need of Graphical Processing Units (GPUs). These GPUs are heavy and power-hungry, thus cannot be placed on drones, also the smaller Nvidia Jetson boards have quite under-powered ARM based CPUs limiting small application as well.

The solution that we propose is to use the Robot Operating System (ROS) in order to transfer the images captured to a separate computer on ground and then do the processing there and transfer the results back using the same architecture. This would allow us use huge computers with great computing power, which would then in turn allow us to use heavier and better performing techniques. This proposal will be limited by the bandwidth of the connection, as of now, this is being circumnavigated by sending images of smaller sizes. In order to further increase the rates of transfer we are only sending compressed images, which provide huge boosts to the processing rates (see table 5.1 for detailed time analysis).

At present we are using the SSD, from the tensorflow object detection API [21] released sometime back.

5.3 Stereo image processing

5.3.1 Stereo Image Capture

The human perception of depth is brought about by the hardly understood brain process of fusing two planar images obtained from slightly different perspective viewpoints. Due to the different viewpoint of each eye, a small horizontal shift exists, called disparity, between corresponding image points in the left and right view images on the retinas. In stereoscopic vision, the objects to which the eyes are focused and accommodated have zero disparity, while objects to the front and to the back have negative and positive disparity, respectively. The differences in disparity are interpreted by the brain as differences in depth Z. [22]

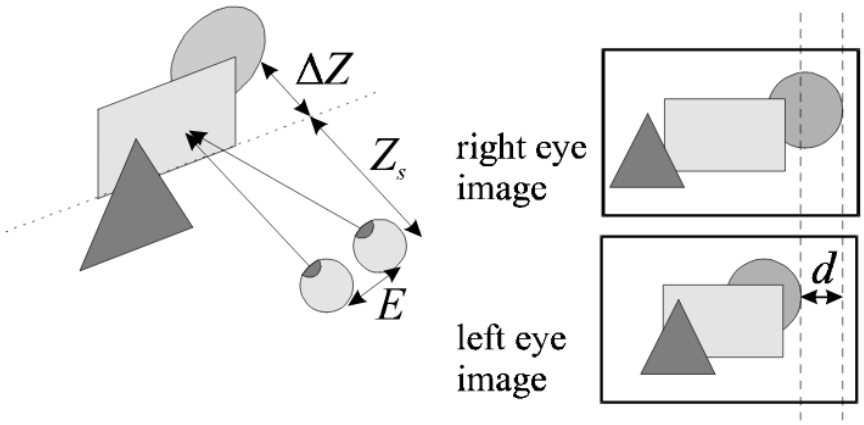

Figure 5.6: Stereoscopic vision, resulting in different disparities depending on depth

In order to be able to perceive depth using recorded images, a stereo camera (for eg. Zed Camera) is required which consists of two cameras that capture two different, horizontally shifted perspective viewpoints. This results in a shift (or disparity) of objects in the recorded scene between the left and the right view depending on their depth. In most cases the interaxial separation or baseline B between the two lenses of the stereoscopic camera is in the same order as the eye distance E (6 to 8 cm). In a simple camera model the optical axes are assumed to be parallel. The depth Z

and disparity d are then related as follows:

$$d = \lambda \frac{B}{\lambda - Z} \tag{5.1}$$

where λ is the focal length of the cameras. A more complicated camera model takes into account the convergence of the camera axes with angle β. The disparity is not only dependent on the depth Z of an object, but also on the horizontal object position X. Furthermore, a converging camera configuration also leads to small vertical disparity components, which are, however, often ignored in subsequent processing of the stereoscopic data.

When recording stereoscopic image sequences, the camera setup should be such that, when displaying the stereoscopic images, the resulting shifts between corresponding points in the left and right view images on the display screen allow for comfortable viewing. If the observer is at a distance Z_s from the screen, then the observed depth Z_{obs} and displayed disparity d are related as:

$$Z_{obs} = Z_s \frac{E}{E - d} \tag{5.2}$$

In the case that the camera position and focusing are changing dynamically, as is the case for instance in stereoscopic television production where the stereo camera may be zooming, the camera geometry is controlled by a set of production rules. If the recorded images are to be used for multiviewpoint stereoscopic display, a larger interaxial lens separation needs to be used, sometimes even up to 1 meter. In any case the camera setup should be geometrically calibrated such that the two cameras capture the same part of the real world scene.

5.3.2 Disparity Estimation

The key difference between planar and stereoscopic images and image sequences is that the latter implicitly contains depth information in the form of disparity between the left and right view images. Not only is the presence of disparity information essential to the ability of humans to perceive depth, disparity can also be exploited for automated depth segmentation of real world scenes, and for compression and interpolation of stereoscopic images or image sequences.

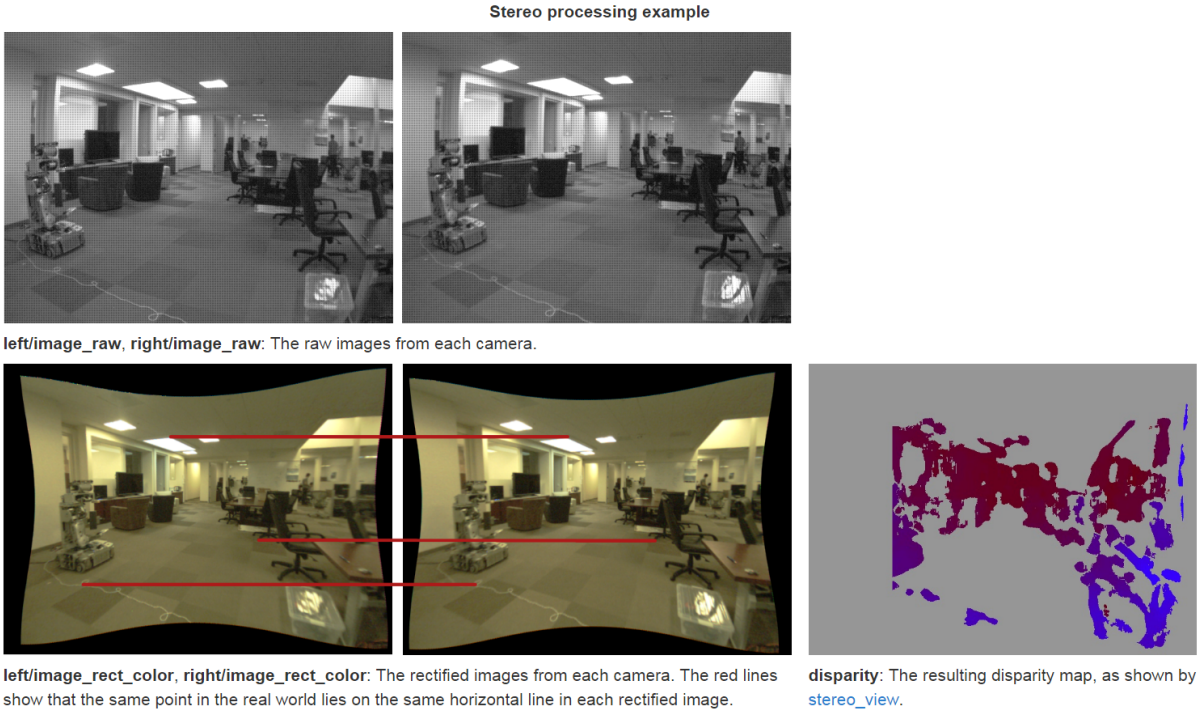

Figure 5.7: Disparity field in the stereo image pair

Disparity estimation is essentially a correspondence problem. The correspondence between the two images can be determined by either matching features or by operating on or matching of small patches of gray values. Feature matching requires as a preprocessing step the extraction of appropriate features from the images, such as object edges and corners. After obtaining the features, the correspondence problem is first solved for the spatial locations at which the features occur, from which next the full disparity field can be deduced by for instance interpolation or segmentation procedures. Feature-based disparity estimation is especially useful in the analysis of scenes for robot vision applications.

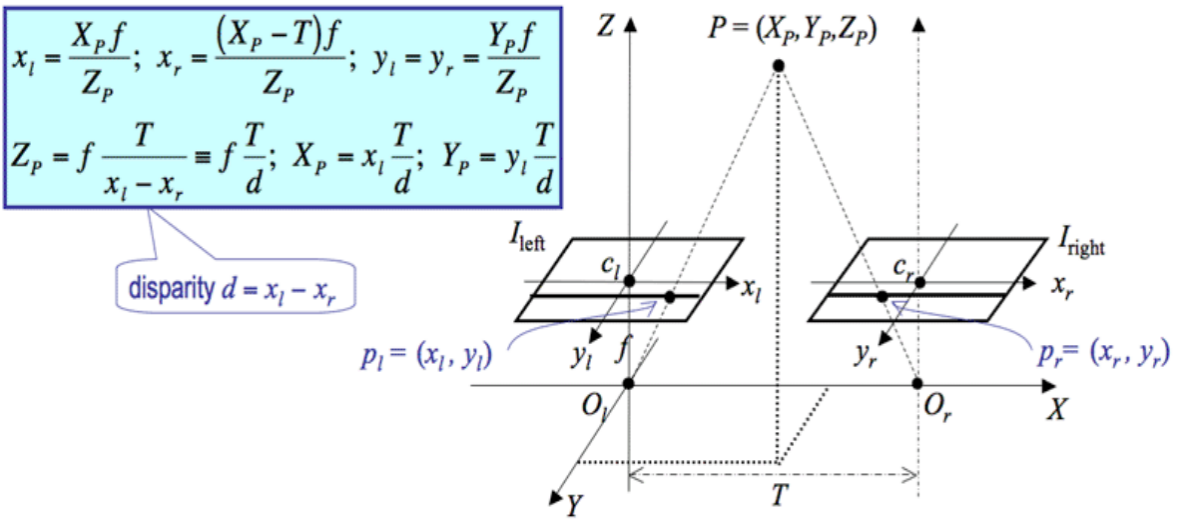

Figure 5.8: Disparity calculation

Most disparity estimation algorithms used in stereoscopic communications rely on matching small patches of gray values from one view to the gray values in the alternate view. The matching of this small patch is not carried out in the entire alternate image, but only within a relatively small search region to limit the computational complexity. Standard methods typically use a rectangular match block of relatively small size (e.g., 8x8 pixels). The relative horizontal shift between a match block and the block within the search region of the alternate image that results in the smallest value of a criterion function used, is then assigned as disparity vector to the center of that match block. Often used criterion functions are the sum of squares and the sum of the absolutes values of the differences between the gray values in the match block and the block being considered in the search region.

In image analysis problems, disparity estimation is often considered in combination with the segmentation of the stereoscopic image pair. Joint disparity estimation and texture segmentation methods partition the image pair into spatially homogeneous regions of approximately equal depth. Disparity estimation in image sequences is typically carried out independently on successive frame pairs. Nevertheless, the need for temporal consistency of successive disparity fields often requires temporal dependencies to be exploited by postprocessing of the disparity fields. If an image sequence is recorded as an interlaced video signal, disparity estimation should be

carried out on the individual fields instead of frames to avoid confusion between motion displacements and disparity.

5.3.3 Working module

Stereo images acquired from the simulation were used to calculate a disparity map, by making use of relative positions of the camera. The specs of the simulated camera are shown in table 5.2. The disparity of features between two stereo images are usually computed as a shift to the left of an image feature when viewed in the right image. In real world applications since stereo images may not always be correctly aligned to allow for quick disparity calculation, images obtained are first rectified for the relative rotations of the cameras. After rectification, the correspondence problem can be solved using an algorithm that scans both the left and right images for matching image features to compute disparity by the normalized correlation.

$$NC = \frac{\sum\sum L(r,c).R(r,c-d)}{\sqrt{(\sum\sum L(r,c)^2).(\sum\sum R(r,c-d)^2)}} \tag{5.3}$$

We make use of the "stereo image proc" package in ROS to calculate and publish the normalized correlation disparity.

Quantity	**Value**
Image Resolution	400x400
h_{fov}	80 degrees
baseline	20 cm
Update rate	30 Hz

Table 5.1: Stereo Camera Specifications

5.4 Distance estimation

5.4.1 Using Stereo camera images

Using disparity image calculated from stereo image matching as well as the object bounding box obtained from the detection node, we estimate the distance of the object from the camera. From the bounding box region extracted from the disparity image, the largest cluster of similar-valued pixels is extracted and identified to be the cluster belonging to the human. A mean value of the pixels belonging to this cluster is calculated, which is used to obtain the depth estimate by the image projection formulae

$$z = \frac{f.T}{V} \tag{5.4}$$

where z is the distance to be estimated, d is the disparity value, f is the focal length of the camera in pixels and T is the baseline in real world units.

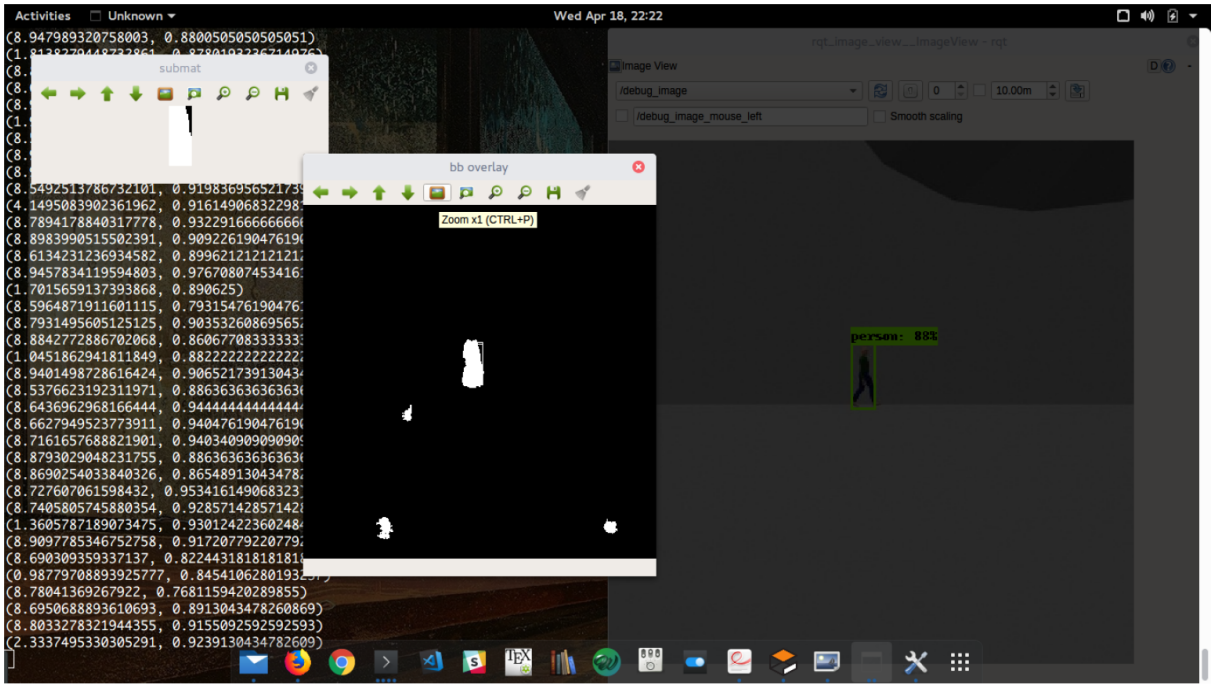

Figure 5.9: Distance estimation results

5.4.2 Using ORB features from monocular camera image

Another monocular camera based approach was developed for depth estimation in the project, using the scaling factors obtained from measuring the spread of ORB features in the image. The algorithm proposed was as follows

- Detect ORB features in the image
- Remove features lying outside bounding box for the image
- match the features in consecutive images and remove unmatched features
- calculate std. deviation of the features in x and y for both t and t+1 image frames
- obtain estimates of rate of change of depth using aspect ratio of bounding boxes
- remove the estimate (x or y) with greater change from previous value

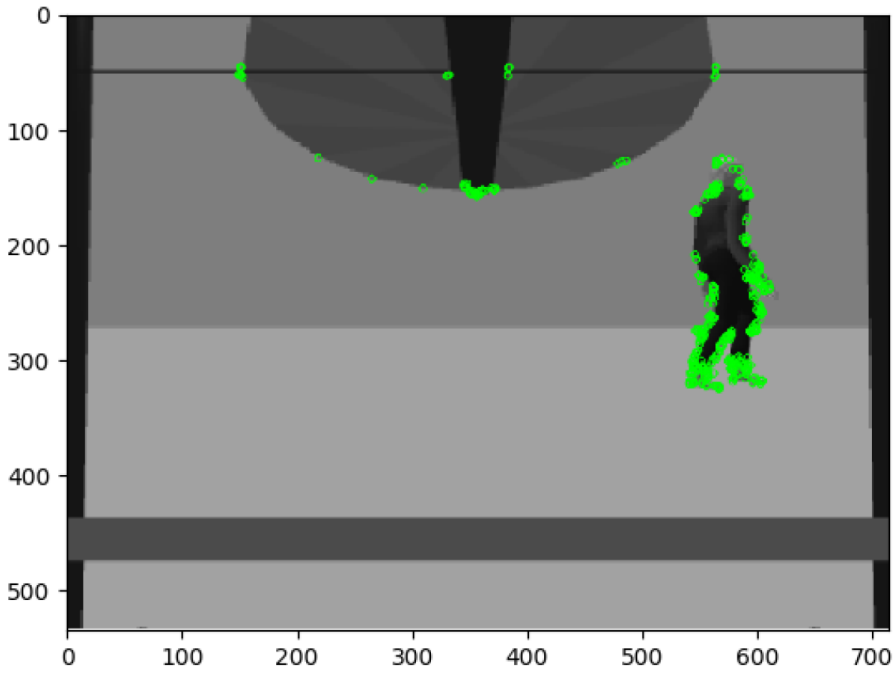

Figure 5.10: ORB features in image frame

The algorithm makes use of the fact that change in depth of an object from camera is inversely proportional to its visible size in the image and directly proportional

to the true size of the object. Hence,

$$\delta z = k.l(\frac{1}{x_t} - \frac{1}{x_{t+1}}) \tag{5.5}$$

where δz is the change in true depth of the object, x_t ,x_{t+1} are sizes of projection in images at t and t+1 frames, l is the true size of object and k is proportionality constant. We estimate the size of the object in the image by the std deviation of the ORB features inside the bounding box in the image. Since this value is susceptible to changes by occlusion, we use the true size of the visible portion as the true size of the object in the formulae. Also adding the assumption the the object will be either occluded in x direction or y direction at a time,

$$\delta z_x = k.(w/L)(\frac{1}{std^t{}_x} - \frac{1}{std^{t+1}{}_x}) \tag{5.6}$$

$$\delta z_y = k.(l/W)(\frac{1}{std^t{}_y} - \frac{1}{std^{t+1}{}_y}) \tag{5.7}$$

where w,l are true sizes of visible portions and W,L are total true sizes of the objects. Hence, quantities w/L, l/W can be estimated by aspect ratio and its inverse of the smaller bounding box in the images.

Finally, the stereo method was chosen since it gave more robust depth estimates.

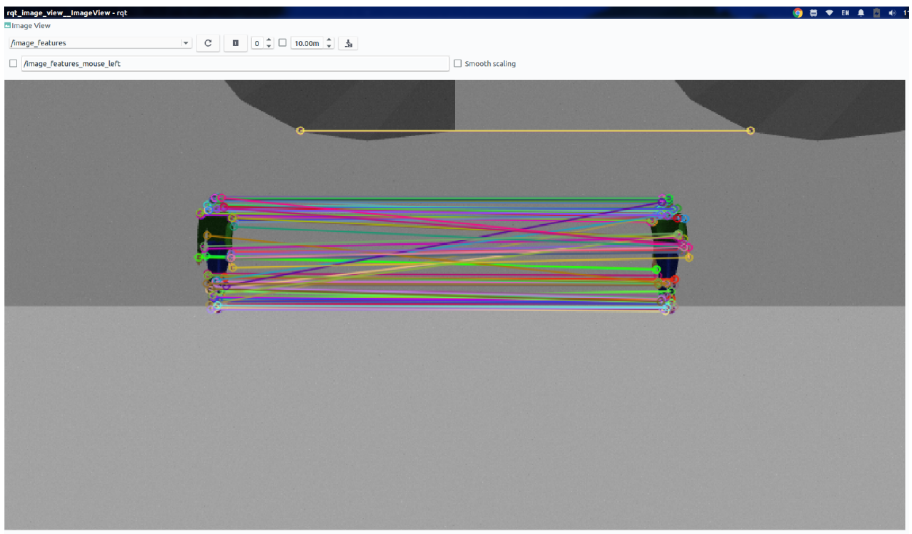

Figure 5.11: Matched ORB features in consecutive frames

5.5 Kalman Filtering

The Kalman filter uses a prediction followed by a correction in order to determine the states of the filter [23]. This is sometimes called predictor-corrector, or prediction-update. The main idea is that using information about the dynamics of the state, the filter will project forward and predict what the next state will be. This can be thought of as a numerical integration technique such as Eulers method or Runge-Kutta. The correction or update part then involves comparing a measurement with what we predict that measurement should be based on our predicted states. The Kalman filtering technique is now discussed in equation format. Starting from some initial state estimate, $\hat{x}_0$, and initial state error covariance matrix, P_0, the predictor-corrector format is applied recursively at each time step, e.g. using a loop. First, the state vector is predicted from the state dynamic equation using

$$\hat{x}_{k|k-1} = F_{k-1}\hat{x}_{k-1} + G_{k-1}u_{k-1} \tag{5.8}$$

where $\hat{x}_{k|k-1}$ is the predicted state vector,$\hat{x}_{k-1}$ is the previous estimated state vector, u is the input vector, and F and G are matrices defining the system dynamics. The subscript $k|k-1$ is read as k given $k-1$ and is a shorthand notation for the state at discrete time k given its previous state at discrete time $k-1$, i.e. this is the prediction of the state using the system model projected forward one step in time. Next, the state error covariance matrix must also be predicted using

$$P_{k|k-1} = F_{k-1}P_{k-1}F_{k-1}^T + Q_{k-1} \tag{5.9}$$

where $P_{k|k-1}$ represents the predicted state error covariance matrix, P_{k-1} is the previous estimated state error covariance matrix, and Q is the process noise covariance matrix. Again,$k|k-1$ is indicating that this is the expected covariance matrix at k based on the system model and the covariance at $k-1$. Once the predicted

values are obtained, the Kalman gain matrix,K_k, is calculated by

$$K_k = P_{k|k-1}H_k^T(H_kP_{k|k-1}H_k^T + R_k)^{-1} \tag{5.10}$$

where $H$is a matrix necessary to define the output equation and R is the measurement noise covariance. The state vector is then updated by scaling the innovation, which is the difference between the measurement of the output, z_k, and the predicted output, $H_k\hat{x}_{k|k-1}$(sometimes called $\hat{y}_{k|k-1}$), by the calculated Kalman gain matrix in order to correct the prediction by the appropriate amount, as in

$$\hat{x}_k = \hat{x}_{k|k-1} + K_k(z_k - H_k\hat{x}_{k|k-1}) \tag{5.11}$$

Similarly, the state error covariance is updated by

$$P_k = (I - K_kH_k)P_{k|k-1} \tag{5.12}$$

where I is an identity matrix.

The estimates of bounding box coordinates obtained from object detection node, as well as the depth estimates obtained from stereo-disparity images and rate of change of depth estimation from ORB features were combined to be fed as measurements for the relative state of the object(human) in pixel coordinates into the kalman filter. The Kalman filter, running at 30 Hz, was employed to smooth out the obtained measurements as well as provide predictions in case the measurement generating nodes failed to provide any measurement. Furthermore, the predictive capability of the kalman filter was also leveraged to provide measurements in case the object to be tracked was occluded, since the measurements will not be generated in such cases as well.

A vanilla Kalman filter can be modelled as

$$\begin{aligned} x_0 &= Gaussian(\mu_0, \Sigma_0) \\ x_{t+1} &= A_t.x_t + b_t + \epsilon^1_{t+1} \\ y_t &= C_t.x_t + d_t + \epsilon^2_t \\ \epsilon^1_t &= Gaussian(0, Q) \\ \epsilon^2_t &= Gaussian(0, R) \end{aligned} \tag{5.13}$$

where x is the model state and y are the measurements. For our case

$$x_t = \begin{bmatrix} x_c \\ y_c \\ z_c \\ v_x \\ v_y \\ v_z \\ a_x \\ a_y \\ a_z \end{bmatrix} y_t = \begin{bmatrix} x_m \\ y_m \\ z_m \\ v_z \end{bmatrix} \tag{5.14}$$

The state-transition and observation matrices for tracking human subject

$$A_t = \begin{bmatrix} 1 & 0 & 0 & T & 0 & 0 & 0.5*T^2 & 0 & 0 \\ 0 & 1 & 0 & 0 & T & 0 & 0 & 0.5*T^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & T & 0 & 0 & 0.5*T^2 \\ 0 & 0 & 0 & 1 & 0 & 0 & T & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & T & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & T \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.15}$$

$$C_t = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{5.16}$$

with b_t and d_t set to 0.

Values of process and measurement noise parameters Q and R are determined from measurement set using Expectation-Maximization, which is a way to find maximum-likelihood estimates for model parameters when your data is incomplete, has missing data points, or has unobserved (hidden) latent variables. It is an iterative way to approximate the maximum likelihood function. It works by choosing random values for the missing data points, and using those guesses to estimate a second set of data. The new values are used to create a better guess for the first set, and the process continues until the algorithm converges on a fixed point. The EM algorithm:

$$Q(\theta|\theta^t) = E_{Z|X,\theta^t}(logL(Z, X, \theta)) \tag{5.17}$$

$$\theta^{t+1} = argmax_\theta Q(\theta|\theta^t) \tag{5.18}$$

Where measurements modelled as X and Z is the latent variable over which distributions are formed. The values of noise parameters (modelled by θ) are estimated during the 2nd step (M-step).

5.6 Object of interest tracking

The implementation of this project in the real world would certainly command the need to use wireless networks for information exchange between the ground based server and the quadrotor. Now, the prime options for communication of this kind would be 4G/5G mobile networks or WiFi, give the short range of WiFi, for long range applications one would move towards the former option. One of the biggest issues which we anticipate this method would have to deal with, would be the occasional frame drop when transferring image data at high rate. In order to deal with this we use a ROS implementation of High Speed Tracking with Kernelized Correlation Filter [24]. Our implementation is based off of code from Tomas Vojir. [25]

Given an initial image patch containing the target, the goal is to learn a classifier to discriminate between its appearance and that of the environment. This classifier can be evaluated exhaustively at many locations, in order to detect it in subsequent frames. Of course, each new detection provides a new image patch that can be used to update the model.

it is tempting to focus on characterizing the object of interest the positive samples for the classifier. However, a core tenet of discriminative methods is to give as much importance, or more, to the relevant environment the negative samples. The most commonly used negative samples are image patches from different locations and scales, reflecting the prior knowledge that the classifier will be evaluated under those conditions.

An extremely challenging factor is the virtually unlimited amount of negative samples that can be obtained from an image. Due to the time-sensitive nature of tracking, modern trackers walk a fine line between incorporating as many samples as possible and keeping computational demand low. It is common practice to randomly choose only a few samples each frame.

Here we use tools to analytically incorporate thousands of samples at different relative translations, without iterating over them explicitly. This is made possible by the discovery that, in the Fourier domain, some learning algorithms actually

become easier as we add more samples, if we use a specific model for translations. These analytical tools, namely circulant matrices, provide a useful bridge between popular learning algorithms and classical signal processing. This tracker is based on Kernel Ridge Re- gression that does not suffer from the curse of ker- nelization, which is its larger asymptotic complexity, and even exhibits lower complexity than unstructured linear regression. Instead, it can be seen as a kernelized version of a linear correlation filter, which forms the basis for the fastest trackers available. The powerful kernel trick is leveraged at the same computational complexity as linear correlation filters.

5.6.1 Fast Kernel Correlation

Even though there are faster algorithms for training and detection, they still rely on computing one kernel correlation each (k^{xx} and k^{xz} respectively). The kernel correlation consists of computing the kernel for all relative shifts of two input vectors. This represents the last standing computational bottleneck, as a naive evaluation of n kernels for signals of size n will have quadratic complexity. However, using the cyclic shift model will allow us to efficiently exploit the redundancies in this expensive computation.

KCF kernel

$$k_{xx'} = exp(-\frac{1}{\sigma^2}(||x||^2 + ||x^{'}||^2 - 2F^{-1}(\tilde{x}.\tilde{x^{'}}))) \tag{5.19}$$

Where x and $x^{'}$ are image patches being compared. This computation can be performed in O(nlog(n)).

The idea behind this method is that given one (or more) labels one should be able to perform online learning in order to discriminate between patches of interest and the background. More details on the original implementation can be easily found. Now since this is a very less computationally involved, the rate of performance goes as up as high as 100 frames per second. This lets us relax up the condition on

the detection pipeline. As for the first labeled frame, we have used the detection bounding box returned after the execution of first frame.

With an interestingly symmetric argument, training with multiple base samples and a single channel can be done in the primal, with only element-wise operations. This follows by applying the same reasoning to the non-centered covariance matrix XTX, instead of XXT. In this case we obtain the original MOSSE filter. For fast element-wise operations we can choose multiple channels (in the dual, obtaining the DCF) or multiple base samples (in the primal, obtaining the MOSSE), but not both at the same time. This has an important impact on time-critical applications, such as tracking. The general case is much more expensive and suitable mostly for offline training applications.

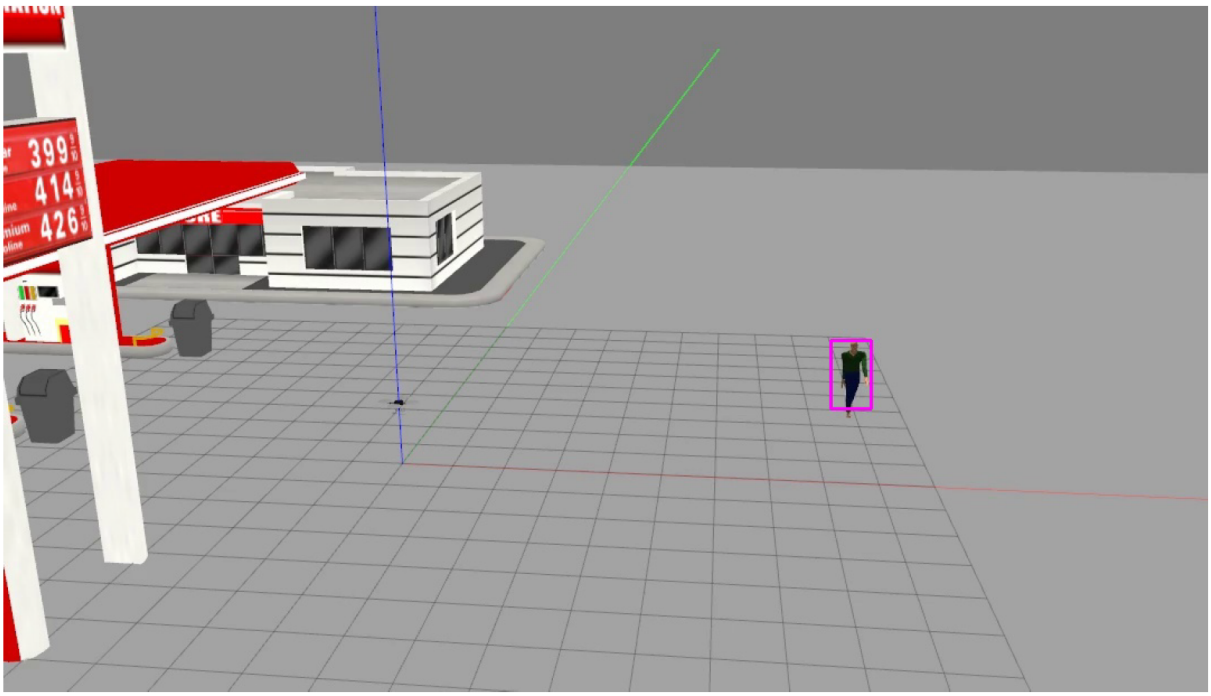

Figure 5.12: KCF Tracked human subject

Chapter 6

Quadrotor based object follow control algorithm

6.1 Quadrotor Model

This section shows the 6-DOF model of a quadrotor based on rigid body dynamics. The derivation of kinematic equations are followed by the derivation of dynamic equations. [26]

6.1.1 Kinematics

For describing the dynamics of a rigid body we use 2 coordinate frames i.e., the inertial and body fixed coordinates. All the physical quantities are transferred between the two coordinate system using the well known Euler angles ($\phi-$ roll, $\theta-$ pitch, $\psi-$ yaw). The following expression relates the velocity of quadrotor the two frames:

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R_b^v \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{6.1}$$

where,

$$R_b^v = \begin{pmatrix} cos\theta cos\psi & sin\phi sin\theta cos\psi - cos\phi sin\psi & cos\phi sin\theta cos\psi + sin\phi sin\psi \\ cos\theta sin\psi & sin\phi sin\theta sin\psi + cos\phi cos\psi & cos\phi sin\theta sin\psi - sin\phi cos\psi \\ -sin\theta & sin\phi cos\theta & cos\phi cos\theta \end{pmatrix} \quad (6.2)$$

Here u,v,w are the velocity coordinates in the body frame and x,y,z are the velocity coordinates in the inertial frame. Similarly, the following expression relates the body rates and Euler angle rates

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & sin\phi sin\theta sec\theta & cos\phi sin\theta sec\theta \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi sec\theta & cos\phi sec\theta \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} \quad (6.3)$$

6.1.2 Dynamics

The dynamical equations for a quadrotor is obtained by applying Newtons second law on a rigid body. The forces acting on a quadrotor can be assumed to be the sum of gravitational force, aerodynamic forces, and propulsion forces. In this work, it is assumed that propulsive force (thrust from the motors) and the gravitational forces are the dominant forces. The aerodynamic force is neglected assuming them to be very small. Transforming the translational dynamics to the inertial frame as follows:

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = R_b^i \begin{pmatrix} 0 \\ 0 \\ \frac{-T}{m} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \quad (6.4)$$

Assuming the quadrotor to be symmetric about x and y axis, the rotational dynamics is given as follows:

$$\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \frac{I_{yy} - I_{zz}}{I_{xx}} qr \\ \frac{I_{zz} - I_{xx}}{I_{yy}} pr \\ \frac{I_{xx} - I_{yy}}{I_{zz}} pq \end{pmatrix} + \begin{pmatrix} \frac{l}{I_{xx}} \\ \frac{m}{I_{yy}} \\ \frac{n}{I_{zz}} \end{pmatrix} \tag{6.5}$$

Note that the cross product terms of the inertia matrix are assumed to be zero. Here, l, m, and n are the components of the externally applied moments known as rolling, pitching, and yawing moments, respectively. All these equations together represent the complete six-DOF of a quadrotor.

6.2 Controller Design

6.2.1 PID control

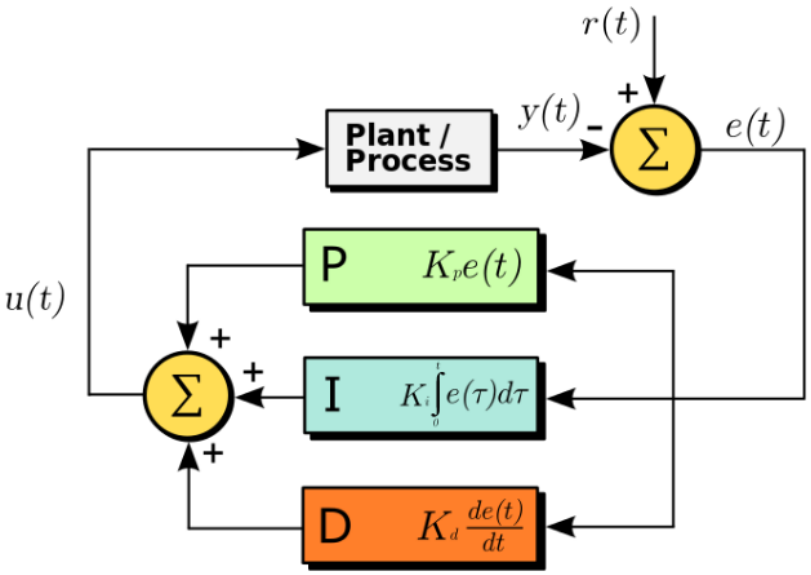

Figure 6.1: PID controller

A PID *(Proportional-Intergral-Derivative)* controller is just another type of feedback controller, based on classical control theory. It its most widely used controller in industrial applications. It is based on continuously calculating the value of an error value between a measured quantity and desired quantity. It then tries to minimize this error over time. The basic control law in PID Controller is :

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt} + K_i \int_0^t e(\tau)d\tau \tag{6.6}$$

where K_p , K_i ,and K_d , all non-negative, denote the coefficients for the proportional, integral, and derivative terms, and $e(t) = r(t) - y(t)$, y(t) is measured process variable and r(t) is desired set point.

6.2.2 Implementation in Quadrotors

6.2.2.1 Outer Loop

As we have already seen, the PID controller calculates the value of an error and then try to minimise it. For quadrotor, PIDs are widely used. Solving the kinematic equations [6.1] and [6.2], we get the following,

$$\begin{aligned} \ddot{x} &= (cos\phi sin\theta cos\psi + sin\phi sin\psi)(-T/m) \\ \ddot{y} &= (cos\phi sin\theta sin\psi + sin\phi cos\psi)(-T/m) \\ \ddot{z} &= (cos\phi cos\theta)(-T/m) + g \end{aligned} \tag{6.7}$$

Assuming small angle approximation, we can actually decouple the system to a fair extent. Then we have,

$$\begin{aligned} \ddot{x} &= \frac{-T}{m}\theta \\ \ddot{y} &= \frac{T}{m}\phi \\ \ddot{z} &= \frac{-T}{m} + g \end{aligned} \tag{6.8}$$

According to PID control law, we write,

$$\begin{aligned} \theta &= K_{p_\theta}(x_d - x) + K_{d_\theta}(\dot{x_d} - \dot{x}) + K_{i_\theta} \int_0^t (x_d - x)dt \\ \phi &= K_{p_\phi}(y_d - y) + K_{d_\phi}(\dot{y_d} - \dot{y}) + K_{i_\phi} \int_0^t (y_d - y)dt \\ \frac{-T}{m} &= K_{p_T}(z_d - z) + K_{d_T}(\dot{z_d} - \dot{z}) + K_{i_T} \int_0^t (z_d - z)dt \end{aligned} \tag{6.9}$$

We have, therefore,

$$\ddot{z} = K_{p_T}(z_d - z) + K_{d_T}(\dot{z_d} - \dot{z}) + K_{i_T}\int_0^t (z_d - z)dt + g \tag{6.10}$$

Solving this, we get the value of (-T/m) and hence,$\ddot{z}$ at every instant of time, thus, we put it into the rest two equation . Thus we have,

$$\begin{aligned}
\ddot{z} &= K_{p_T}(z_d - z) + K_{d_T}(\dot{z_d} - \dot{z}) + K_{i_T}\int_0^t (z_d - z)dt + g \\
\ddot{x} &= (\ddot{z} - g)(K_{p_\theta}(x_d - x) + K_{d_\theta}(\dot{x_d} - \dot{x}) + K_{i_\theta}\int_0^t (x_d - x)dt) \\
\ddot{y} &= (g - \ddot{z})(K_{p_\phi}(y_d - y) + K_{d_\phi}(\dot{y_d} - \dot{y}) + K_{i_\phi}\int_0^t (y_d - y)dt)
\end{aligned} \tag{6.11}$$

Solving these we get, instantaneous x,y,z with respect to time by choosing the appropriate values of K_p , K_d and K_i

6.2.2.2 Inner Loop

Since now, we have our ϕ_d , θ_d i.e.desired value of ϕ and θ respectively, from the output of outer loop, we can proceed to Inner Loop. The value of ψ_d is assumed to be constant and is provided by Mission Planner. We have,

$$\begin{aligned}
\ddot{\phi} &= \frac{L}{I_{xx}}\theta \\
\ddot{\theta} &= \frac{M}{I_{yy}}\phi \\
\ddot{\psi} &= \frac{N}{I_{zz}}
\end{aligned} \tag{6.12}$$

Now,again

$$\begin{aligned}
L &= K_{p_L}(\phi_d - \phi) + K_{d_L}(\dot{\phi_d} - \dot{\phi}) + K_{i_L}\int_0^t (\phi_d - \phi)dt \\
M &= K_{p_M}(\theta_d - \theta) + K_{d_M}(\dot{\theta_d} - \dot{\theta}) + K_{i_M}\int_0^t (\theta_d - \theta)dt \\
N &= K_{p_N}(\psi_d - \psi) + K_{d_N}(\dot{\psi_d} - \dot{\psi}) + K_{i_N}\int_0^t (\psi_d - \psi)dt
\end{aligned} \tag{6.13}$$

Solving for ϕ, θ, ψ by putting the values of L, M, N , we get differential equations. Solving them, we get our instantaneous $\phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}$ and by this we get our L,M and N.

Further by putting the values of L,M,N in a coupled system, we now solve for the following matrices simultaneously to get p,q,r, ϕ, θ and ψ .

$$\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \frac{I_{yy} - I_{zz}}{I_{xx}} qr \\ \frac{I_{zz} - I_{xx}}{I_{yy}} pr \\ \frac{I_{xx} - I_{yy}}{I_{zz}} pq \end{pmatrix} + \begin{pmatrix} \frac{l}{I_{xx}} \\ \frac{m}{I_{yy}} \\ \frac{n}{I_{zz}} \end{pmatrix} \tag{6.14}$$

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & sin\phi sin\theta sec\theta & cos\phi sin\theta sec\theta \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi sec\theta & cos\phi sec\theta \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} \tag{6.15}$$

6.2.3 Control Node

The rectified control input is obtained from Kalman filter/Tracking node. The input is fed into a hierarchical controller, highest level of which uses a cascade of two PID controllers for the horizontal movement and two separate PID controllers for the yaw rate and vertical velocity. The velocity outputs are used for feeding into the lower level controller which calculates the necessary forces and torques acting on the body. The torques and forces hence generated are then applied on the simulation to make the quadrotor move. The low level control is is handled by Gazebo plugins, and hence we tune the high level controller.

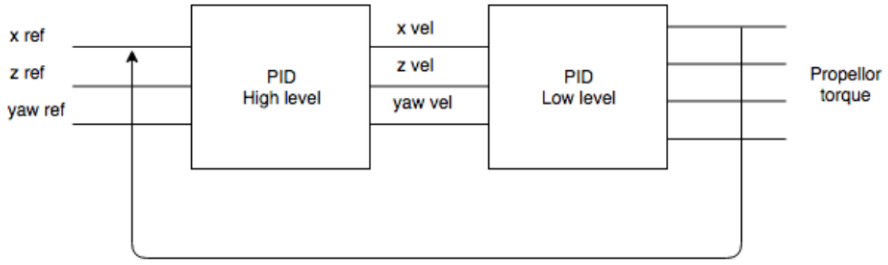

Figure 6.2: Cascaded controller overview

Chapter 7

Environment simulations and results

Developing the pipeline requires a simplified testbed which simulates the real world to a certain extent, in order for the developed pipeline to be robust and be directly adaptable to real world use cases with minimum modifications. We have used Gazebo 8.0 for simulation purposes, since it has direct integration with Robot Operating System (ROS). A custom world was created for the purpose, consisting of a human model with waypoints set to walk in a square, and a quadcopter with stereo camera mounted on it. The images acquired by the quadcopter were published onto ROS topics so that they are accessible just like a real camera image. In order to test for occlusions we have also included a building. The simulation uses ros-control package for realistic quadrotor controls, this package is also used on real world robot applications.

7.1 Experiments and results

7.1.1 Experiments

A gazebo simulation is used for all of the experiments. The simulation consists of a quadrotor and actor based on humans moving on certain trajectories, which at

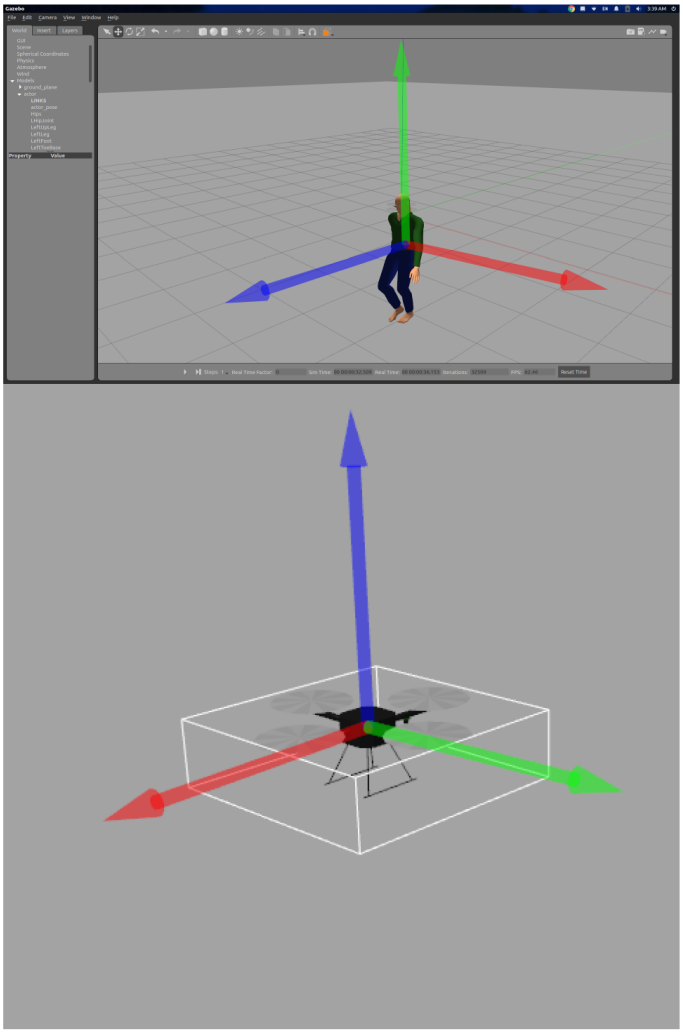

Figure 7.1: Simulation models of quadrotor and human subject

the moment are fixed. The quadrotor used is *hector-quadrotor* and for capturing images the onboard camera is used, the resolution of capture is 320x240. Now, the reason methods discussed before works, which are based on feature point detection, is when testing for detection of ORB features on the obtained image we got 400 feature points.

Images captured are sent to GPU server on which a tensorflow API is used to perform detection using SSD, this is done to simulate the real world implementation of the pipeline. The frequency of the camera was set to 15Hz as the fastest detection was being done at 14Hz and feeding in more frames would only consume bandwidth of the connection used to transfer images between the GPU server and simulation.

Support was added for more complex trajectories of the human subject such

as moving in a square and with varying speeds and turn rates. This allows the simulation to better model real world scenario. Also, this increased a challenge for quadrotor since the subject no longer permanently stayed inside the camera field of view. This also helped avoid local optima states for the quadrotor where it could keep the subject at center by just changing the yaw values.

Depth estimation portions were worked upon both in using stereo camera as well as monocular version, since the previous method of estimating the distance by the area of the bounding box was unreliable. The area based estimate changed quadratically with distance, voiding the linear requirement of the simple control, as well as changed drastically in case of occlusion or if the human subject was at an edge of the frame.

Kalman filter was implemented for high rate control input generation as well as occlusion avoidance. This helped take care of the issue of missing detections in hard-to-detect frames as well as smoothed out the detections, aiding in easier control. It also provided an added functionality of ability to incorporate occlusion/ out-of-frame cases.

Tracking node was developed for object detection in crowded environments. KCF was incorporated into the pipeline along with kalman filter to provide better estimates since it also uses information directly from the frames unlike kalman filter. This provided an added functionality of ability to incorporate cases with multiple moving subjects.

7.1.2 Specifications

Quadrotor specifications: Hector quadrotor gazebo model

Inertial Mass Value: 1.477 units

Onboard sensors(for implementation on a real quadrotor):

- 3 x gyroscopes
- 3D accelerometer
- 3D magnetometer
- Vertical distance (ultrasound), 16.66 Hz sample rate

Figure 7.2: Experimental setup in Gazebo

- Power: LiPo batteries, 6 cells, 22.2 V, 222.0 Wh, 10000 mAh

Base Station capabilities(for implementation on real quadrotor) :

- WiFi connectivity: 2.4 GHz and 5.0 GHz spectra, 100Mbps, TCP/IP support
- Other connectivity: 3G, (2100/850/900 MHz), 4G LTE

Mounted camera specifications: Generic stereo camera

- Update rate: 30 Hz
- Resolution: 400 x 400
- Image format: RGB
- Horizontal field of view: 80°

Processing time and requirements for image transfer, detection:

Devices	Image capture	Detection	Uncompressed transfer	Compressed
Intel i5-3250, No-GPU	34ms	130 ms	NA	NA
Nvidia GeForce 960M	34ms	60ms	NA	NA
Nvidia Titan X	34MS	5ms	500ms	10ms

Table 7.1: Time Analysis

System requirements for running object detection node:

- CUDA Toolkit 8.0.
- The NVIDIA drivers associated with CUDA Toolkit 8.0.
- GPU card with CUDA Compute Capability 3.0 or higher.

7.1.3 Results

Detailed time based analysis can be obtained in Table 7.1 for various submodules in the pipeline.

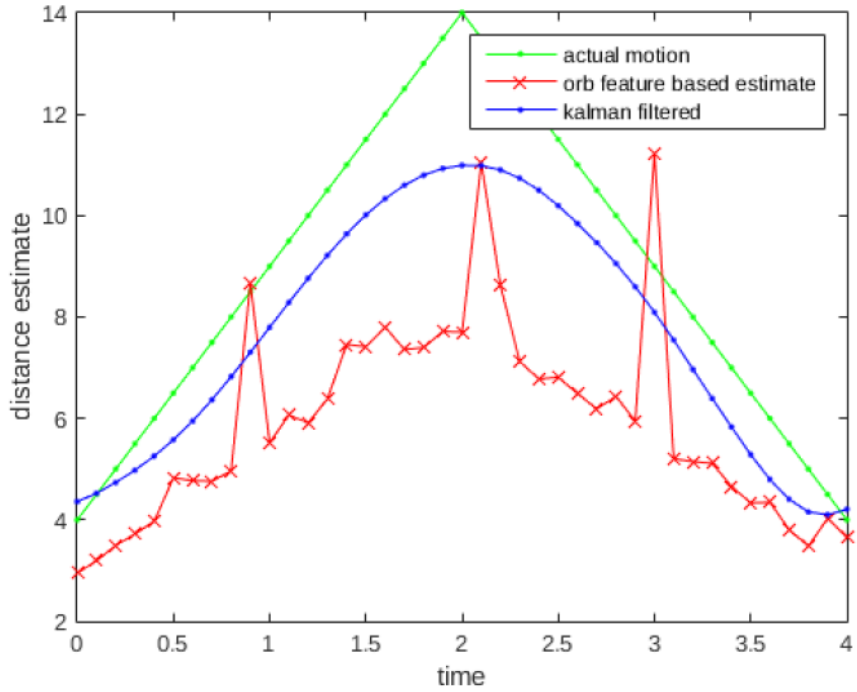

Figure 7.3: Results of distance estimation by Kalman filter as well as orb feature method against ground truth

The results were recorded in rosbag files which are file formats for storing ROS message data. These subscribe to a certain specific topics and can also be played back to the same topics.The rosbag files are further changed into .mat files for working in MATLAB using a bagReader module available in MATLAB. Then the data is used for plotting the results in MATLAB graphs.

The corners of the human trajectory is made erratic to simulate the uncertain nature of human movement. The quadrotor is set to follow the human keeping different distance threshold to check the effectiveness at 3m, 5m, and 10m. The effect of varying height is also tested as the quadrotor is set at 1.5m, 3m and 5m above the ground. A different path with different velocity is also tested on.

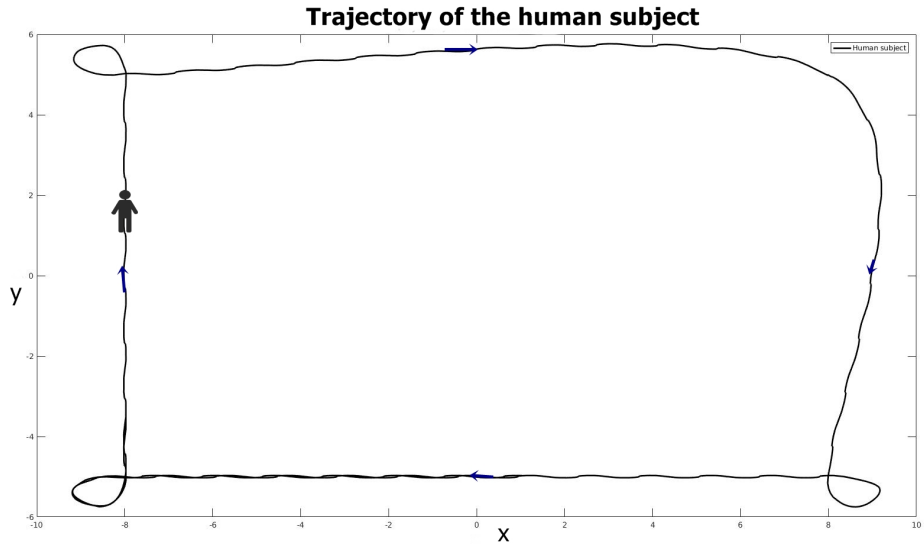

Figure 7.4: Trajectory of the human subject

7.1.3.1 Results for 3m distance, 1.5m height

The results shows the quadrotor starting from origin and aligning itself to follow the human subject trying to keep the distance required.

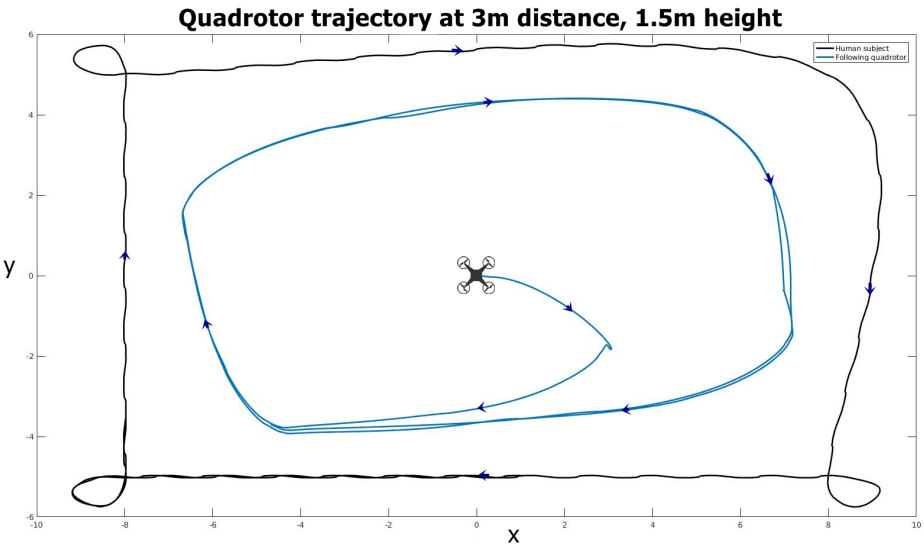

Figure 7.5: Quadrotor path for 3m distance, 1.5m height

The distance vs time graph shows the deviation in distance is around average of 3m.

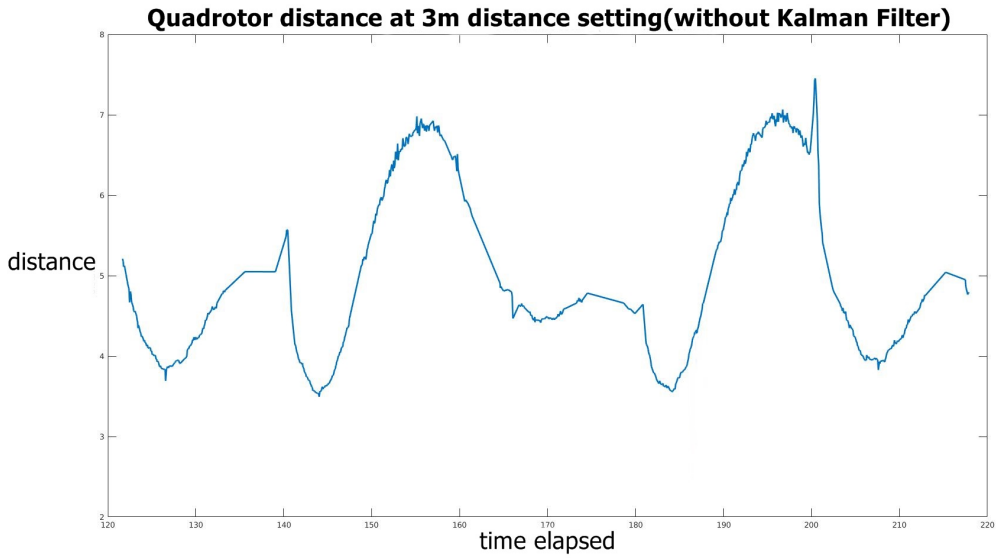

Figure 7.6: Distance of quadrotor for 3m setting, 1.5m height

7.1.3.2 Results for 5m distance, 1.5m height

The result shows the quadrotor is able to effectively track the subject from a distance of 5m.

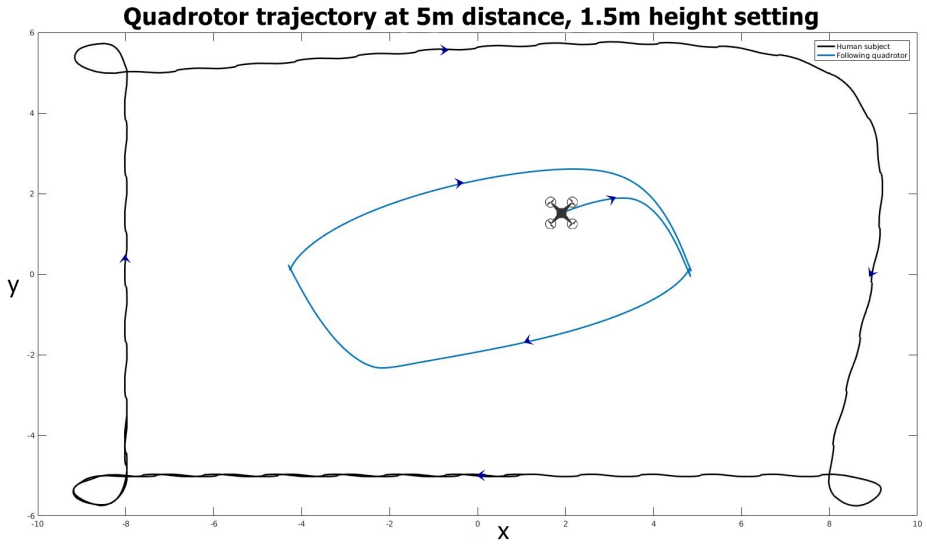

Figure 7.7: Quadrotor path for 5m distance, 1.5m height

The distance vs time graph shows the deviation from the set point for distance is at an average of 3m without applying kalman filter.

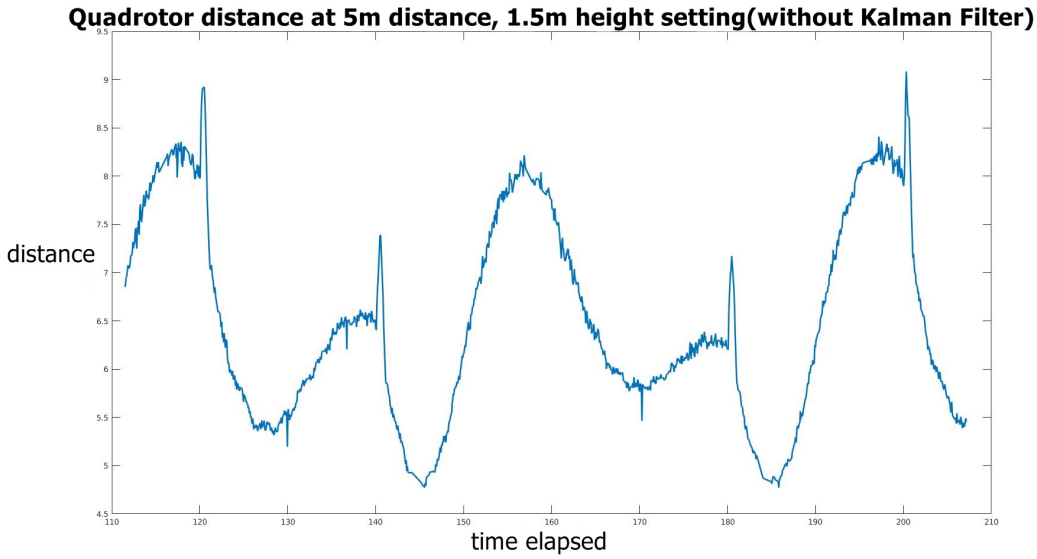

Figure 7.8: Distance of quadrotor for 5m setting, 1.5m height, without Kalman filter

While on using Kalman filter, the movement of the quadrotor is smoothed out, and also the average deviation from the set distance is reduced.

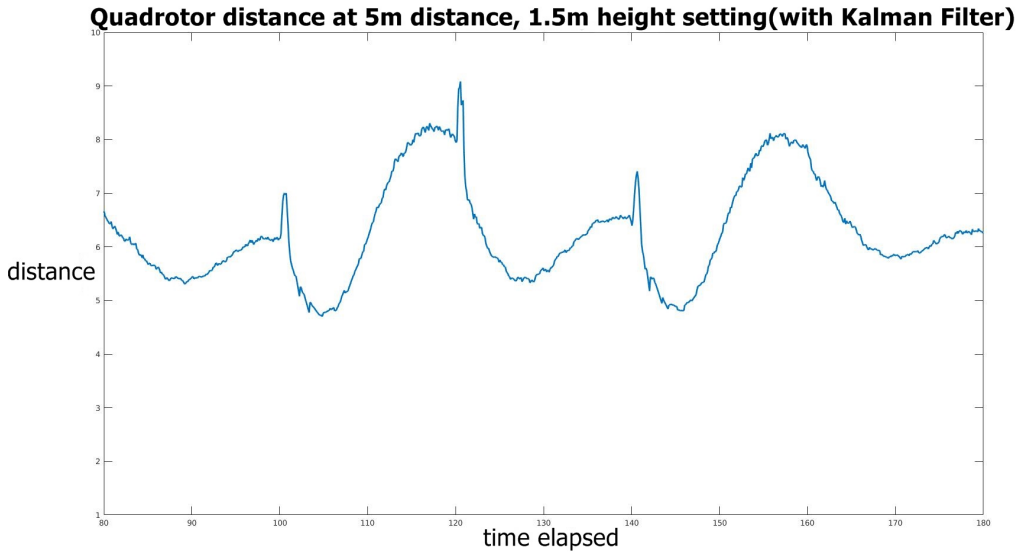

Figure 7.9: Distance of quadrotor for 5m setting, 1.5m height, with Kalman filter

Following graph shows the midpoints of the detected bounding box along the whole time the quadrotor takes to complete the trajectory. The cluster of points shows that deviation in x and y direction is 30 and 20 pixels respectively, which is quite small.

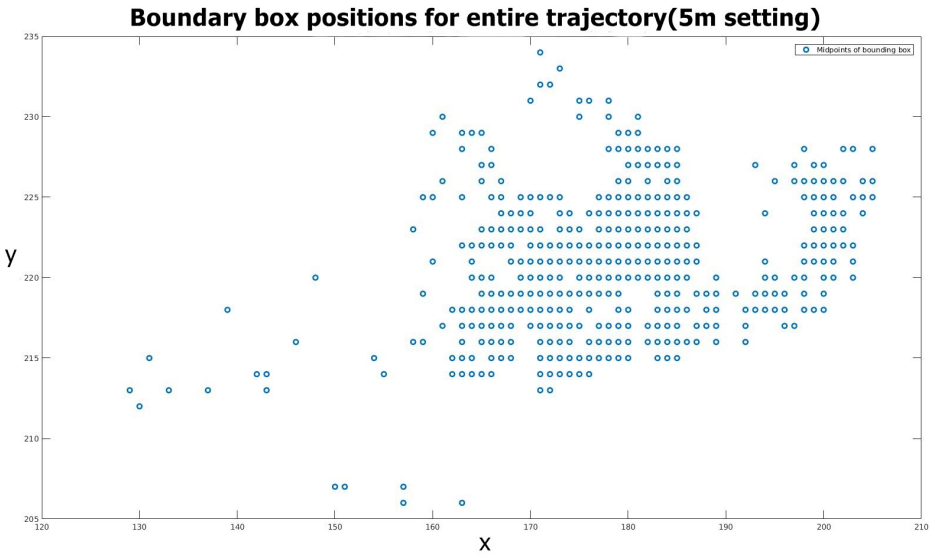

Figure 7.10: Bounding box positions for 5m distance, 1.5m height

The confidence vs time graph shows the confidence level lies between 0.5 to 0.9. So the prediction is very much accurate about detecting human subject.

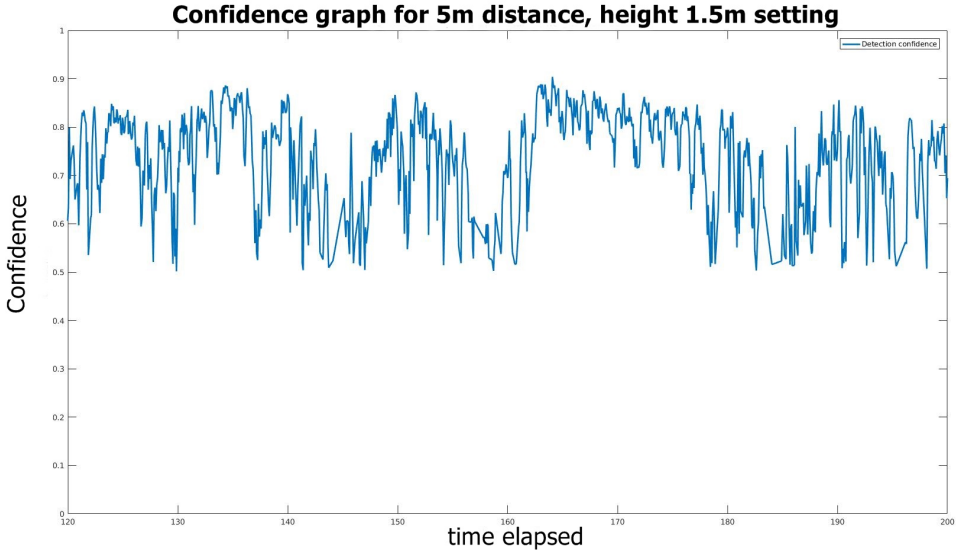

Figure 7.11: Confidence graph for 5m distance, 1.5m height

7.1.3.3 Results for 5m distance, 3m height

The graphs show that the change in height has very negligible effect on the tracked path followed by the quadrotor.

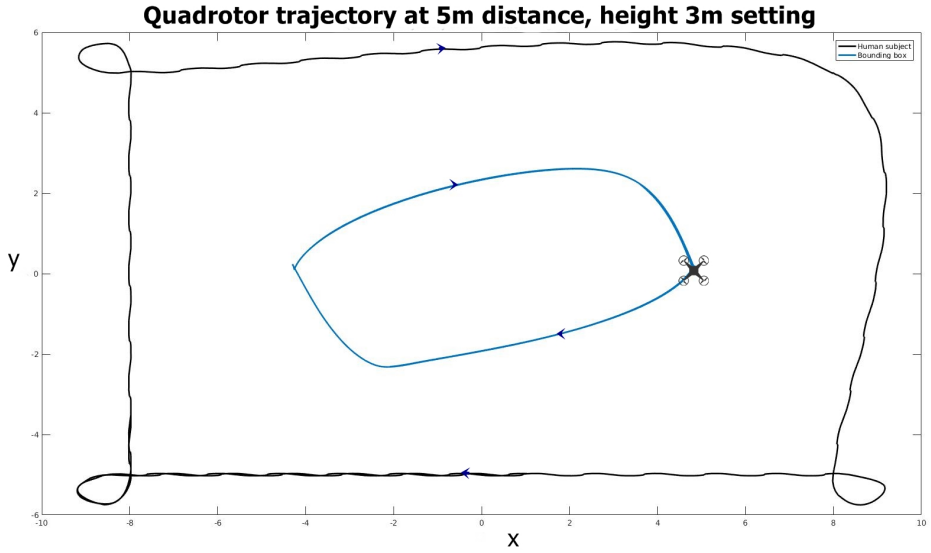

Figure 7.12: Quadrotor path for 5m distance, 3m height

The distance vs time graph shows the deviation from the set point for distance is at an average of 3m.

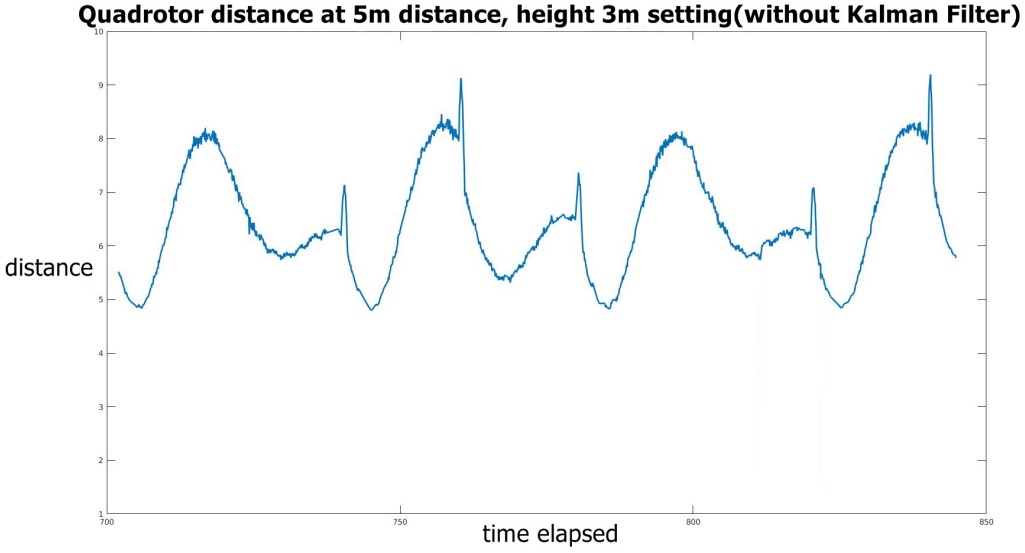

Figure 7.13: Distance of quadrotor at 5m distance, 3m height

The cluster of points shows that deviation in x and y direction is 30 and 20 pixels respectively, which is similar to previous case.

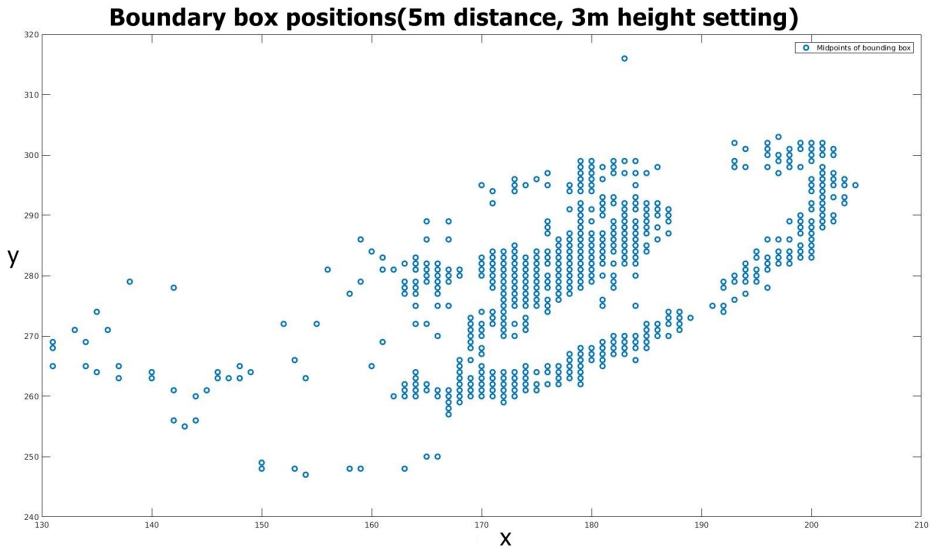

Figure 7.14: Bounding box positions for 5m distance, 3m height

The confidence vs time graph shows the confidence level lies between 0.5 to 0.9.

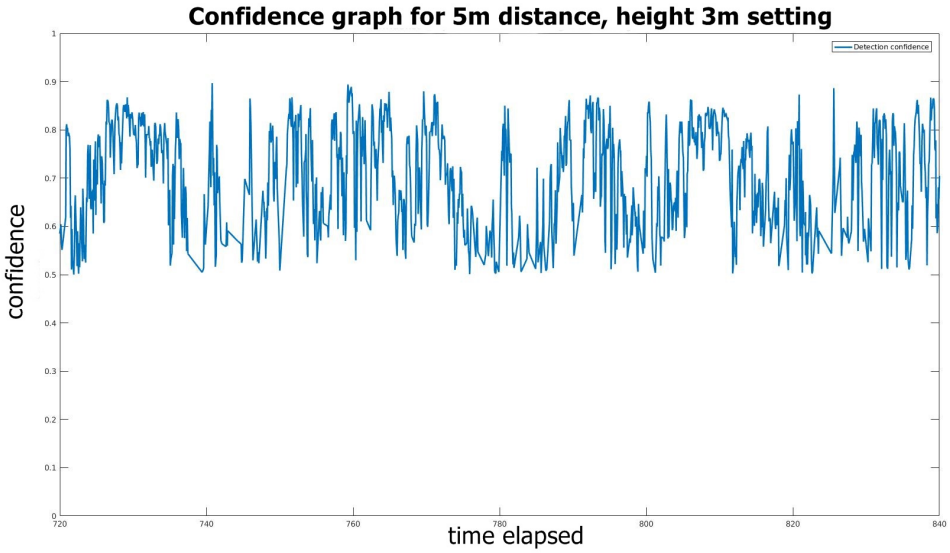

Figure 7.15: Confidence graph for 5m distance, 3m height

7.1.3.4 Results for 10m distance, 1.5m height

The results for 10m distance setting showed some erratic results mainly due to the large distance setting. The quadrotor's yaw control took over in this scenario as the human subject can be seen for a greater field of view.

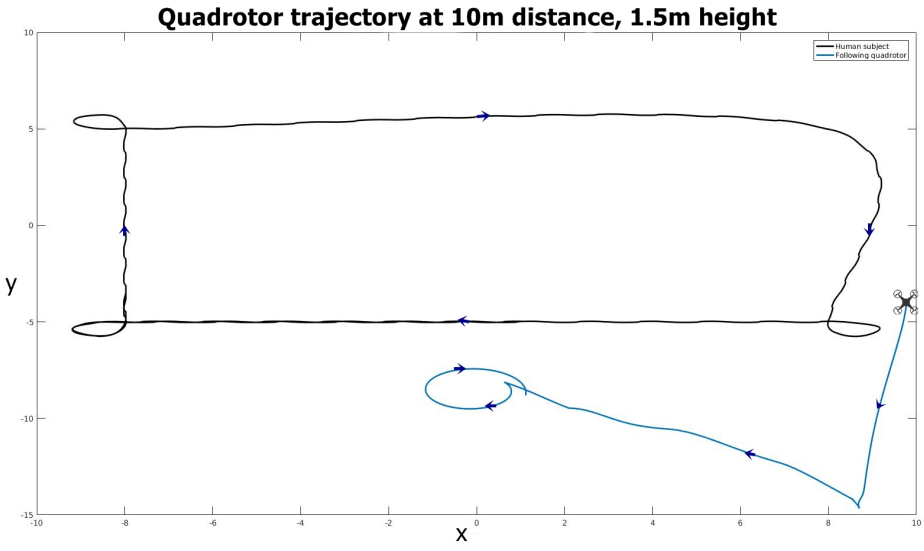

Figure 7.16: Quadrotor path for 10m distance, 1.5m height

The distance vs time graph shows irregular results cause the initial quadrotor position was outside the human trajectory. The quadrotor required some time to start tracking proper track, causing the irregularity.

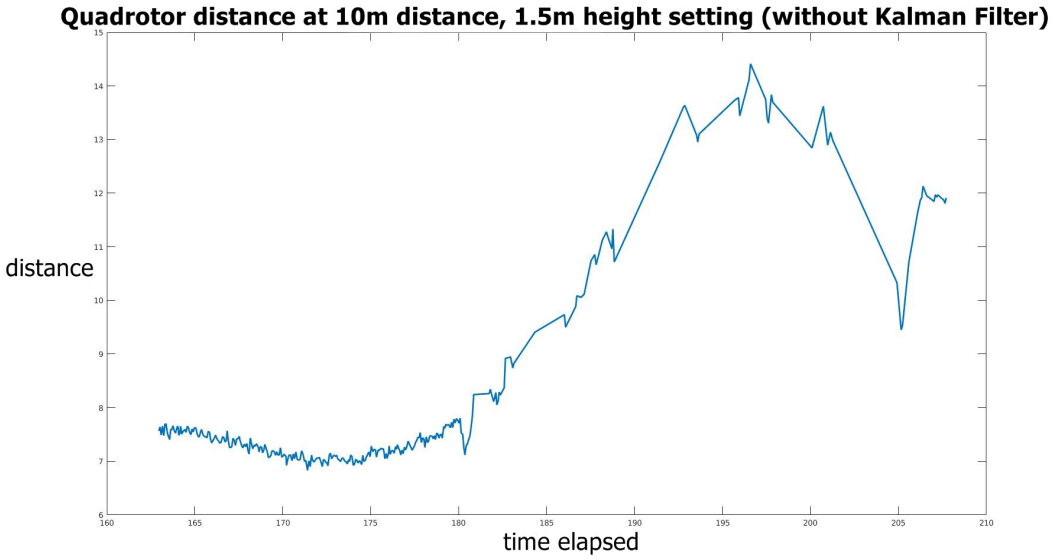

Figure 7.17: Distance of quadrotor at 10m distance, 1.5m height

The bounding box graph shows very sparse amount of points within the time frame. This clearly indicates that the detection was very less at this distance setting.

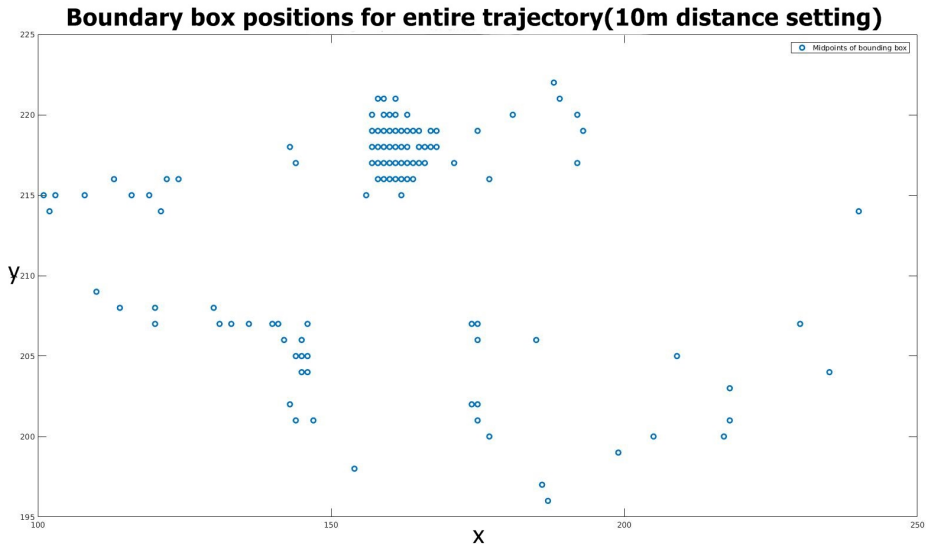

Figure 7.18: Bounding box positions for 10m distance, 1.5m height

The confidence graph also verifies the fact that the detection was very less, causing the jumps and shifts in the graph. Inspite of the meagre amount of detection,the

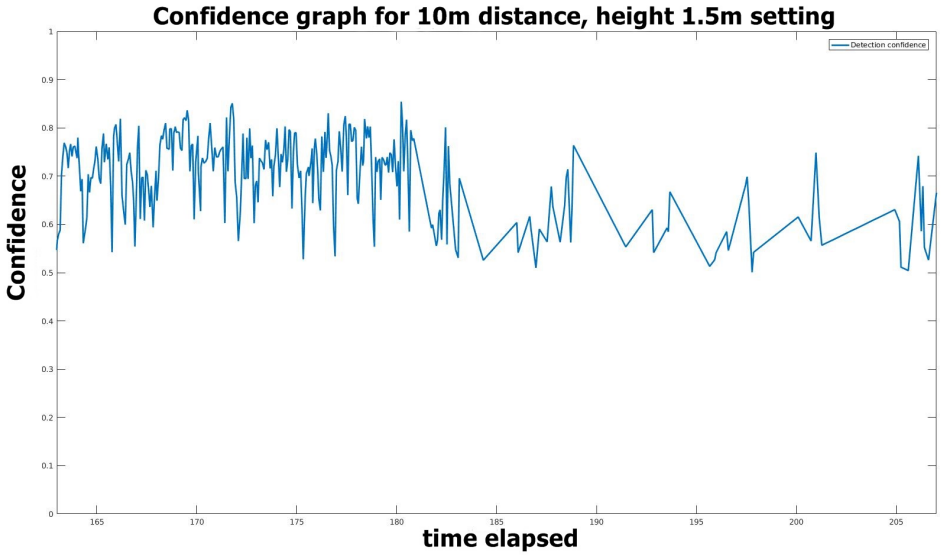

Figure 7.19: Confidence graph for 10m distance, 1.5m height

algorithm was able to effectively track the human subject, confirming the feasibility of the method.

7.1.3.5 Results for different human trajectory path at 3m distance, 2x velocity

This scenario uses a different path for the human to follow. Note that the sudden changes in the path was ignored by the quadrotor with the help of kalman filter, making a smooth trajectory for itself, while keeping the detection rate intact.

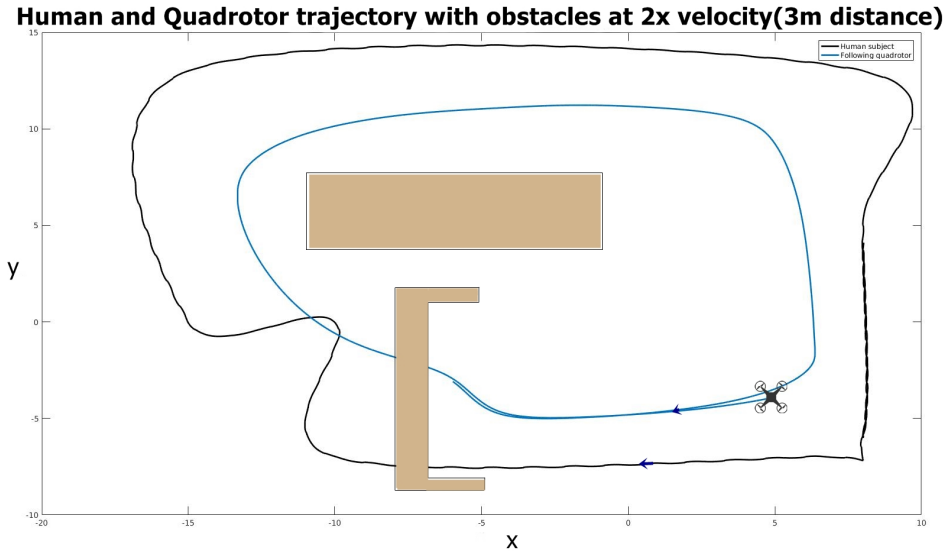

Figure 7.20: Quadrotor path for different trajectory, 2x velocity

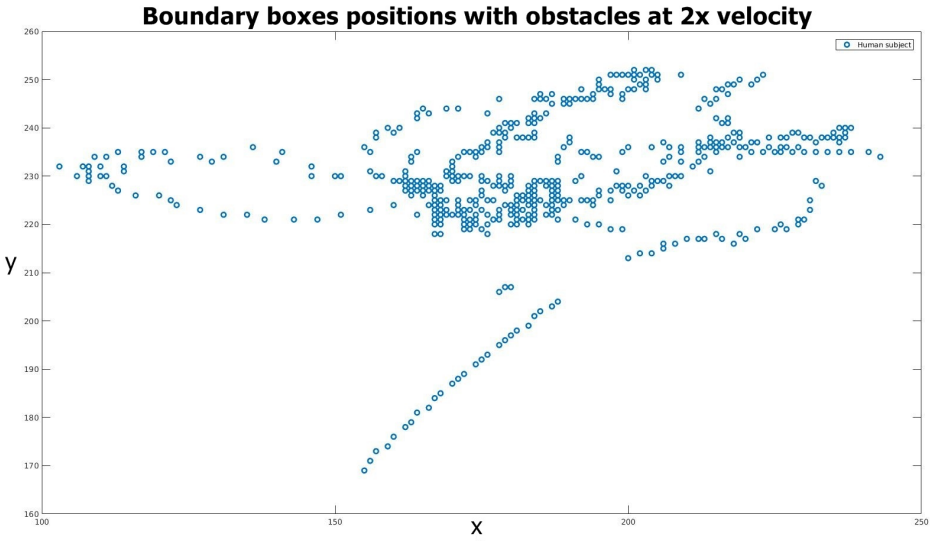

Figure 7.21: Bounding box positions for for different trajectory, 2x velocity

Chapter 8

Conclusion and future work

We developed a system for detecting and following a human subject autonomously via a quadrotor with only vision based techniques. The payload over the quadcopter was kept to a minimum by offloading high computation tasks to a remote server with a GPU. The system was made robust to occlusion and/or presence of other subjects via KCF tracking and Kalman filtering. The proposed system was tested to be able to function in a Gazebo simulation.

Limitations of the proposed system include the requirement of a high speed and bandwidth connection to the remote server machine to transport images and similar data. Hence, functioning over WiFi offered us suboptimal results. Another limitation is that the initial frame seen by the quadcopter should only contain the target human subject, since the system cannot decide between different subjects in the first frame. Also, the system is not yet robust to heavy jerks in the camera video feed, as well as it fails when it loses the sight of the human target for longer periods of time.

8.1 Future Work

"One of the virtues of working on simulation is the lack of real world problems one has to face irrespective of how accurately the simulation models the real world, since the real world is just broken." Besides the implementation on a real world quadrotor, the following things can be worked on as extensions to this project:

- The model can be edited to include a gimbaled camera which would reduce the adverse effect on the camera image due to the motion of the quad.
- Exploration of trajectory generation techniques and maybe a move towards optimal trajectory- generation.
- Other filters from the kalman family can be explored.

Bibliography

[1] J.-P. Su and K.-H. Hsia, "Height estimation and image tracking control of an indoor quad-rotor craft via multi-vision systems," *International Journal of Computer, Consumer and Control (IJ3C)*, vol. 2, no. 4, 2013.

[2] T. Naseer, J. Sturm, and D. Cremers, "Followme: Person following and gesture recognition with a quadrocopter," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 624–630, IEEE, 2013.

[3] T. Müller and M. Müller, "Vision-based drone flight control and crowd or riot analysis with efficient color histogram based tracking," in *Airborne Intelligence, Surveillance, Reconnaissance (ISR) Systems and Applications VIII*, vol. 8020, p. 80200R, International Society for Optics and Photonics, 2011.

[4] A. Qadir, J. Neubert, W. Semke, and R. Schultz, "On-board visual tracking with unmanned aircraft system (uas)," in *Infotech@ Aerospace 2011*, p. 1503, 2011.

[5] Y. Imamura, S. Okamoto, and J. H. Lee, "Human tracking by a multi-rotor drone using hog features and linear svm on images captured by a monocular camera," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, pp. 8–13, 2016.

[6] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 25, no. 5, pp. 564–577, 2003.

[7] S.-K. Weng, C.-M. Kuo, and S.-K. Tu, "Video object tracking using adaptive kalman filter," *Journal of Visual Communication and Image Representation*, vol. 17, no. 6, pp. 1190–1208, 2006.

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

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

[10] M. Achtelik, T. Zhang, K. Kuhnlenz, and M. Buss, "Visual tracking and control of a quadcopter using a stereo camera system and inertial sensors," in *Mechatronics and automation, 2009. icma 2009. international conference on*, pp. 2863–2869, IEEE, 2009.

[11] R. Barták and A. Vykovskỳ, "Any object tracking and following by a flying drone," in *Artificial Intelligence (MICAI), 2015 Fourteenth Mexican International Conference on*, pp. 35–41, IEEE, 2015.

[12] C.-T. Dang, H.-T. Pham, T.-B. Pham, and N.-V. Truong, "Vision based ground object tracking using ar. drone quadrotor," in *Control, Automation and Information Sciences (ICCAIS), 2013 International Conference on*, pp. 146–151, IEEE, 2013.

[13] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 7, pp. 1409–1422, 2012.

[14] B. D. Lucas, T. Kanade, *et al.*, "An iterative image registration technique with an application to stereo vision," 1981.

[15] A. Lee, "Comparing deep neural networks and traditional vision algorithms in mobile robotics," *Swarthmore College*, 2015.

[16] A. Sachan, "Guide to object detection using deep learning: Faster r-cnn,yolo,ssd." `http://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/`.

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

[18] "Robot operating system." `https://en.wikipedia.org/wiki/Robot_Operating_System`.

[19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[20] "Object detection using single shot multibox detector." `http://cv-tricks.com/object-detection/single-shot-multibox-detector-ssd/`.

[21] Tensorflow, "Tensorflow object-detection api." `https://github.com/tensorflow/models/tree/master/research/object_detection`.

[22] R. L. Lagendijk, R. E. Franich, and E. A. Hendriks, "Stereoscopic image processing," *Delft University of Technology, Electrical Engineering, Delft, The Netherlands*, 2002.

[23] M. B. Rhudy, R. A. Salguero, and K. Holappa, "A kalman filtering tutorial for undergraduate students," *International Journal of Computer Science & Engineering Survey (IJCSES)*, vol. 8, pp. 1–18, 2017.

[24] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 583–596, 2015.

[25] T. Vojir, "Open source implementation of high speed tracking with kernelized correlation filters." https://github.com/vojirt/kcf.

[26] "Controlling of a single drone: Hovering the drone during flight modes." se.wtb.tue.nl/~lefeber/do_download_pdf.php?id=159.