ADAPTIVE LONG TERM TRACKING AND AUTONOMOUS FOLLOWING USING STEREO-CAMERA OF AN UNMANNED AERIAL VEHICLE WITH COLLISION AVOIDANCE

by

Vignesh Babu

Submitted in partial fulfillment of the
requirements for the degree of
Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
July 2015

© Copyright by Vignesh Babu, 2015

Dedicated to each and every wonderful being
on planet earth and beyond

Table of Contents

List of Tables

List of Figures

Abstract

Unmanned aerial vehicles commercially called quadcopters or drones have become increasingly popular over recent years, delving into wide range of fields from medicine for providing immediate health care or in agriculture for locating damaged crops using special sensors to being used in quarries for 3d mapping.

We focus on the application of drones in adaptive long term tracking of an object-of-interest and following it with necessary collision avoidance. For this we have implemented a tracking framework called TLD, employing an integrated stereo camera on-board the commercial drone Spiri as the sensor to perform long-term tracking of a target object and use the depth map generated from the disparity of the stereo camera to maintain necessary distance from the target. This is built over the ROS framework. We examine and demonstrate this design in real-time on a commercial drone with monocular camera and in simulation on a model drone integrated with stereo camera. We further refined the tracking process by remodeling TLDs tracker to work with SIFT features supplemented by depth information.

We present the evaluation results to show the improvements achieved by our algorithm to autonomously maneuver the drone in making smooth and rapid transitions and then provide comparisons to show improved tracking resilience against modest change in object appearance immediately following system initialization.

Glossary

DOF Degree of Freedom.

DOG Difference of Gaussian.

GUI Graphical User Interface.

HCI Human Computer Interaction.

HRI Human Robot Interaction.

IMU Inertial Measurement Unit.

KLT Kanade-Lucas Tracker.

MAV Micro Aerial Vehicle.

NCC Normalized Cross-Correlation.

OpenCV Open source computer vision library.

ROS Robot Operating System.

RQT ROS based QT framework.

SIFT Scale Invariant Feature Transform.

TLD Tracking-Learning-Detection.

UAS Unmanned Aerial System.

UAV Unmanned Aerial Vehicle.

UGV Unmanned Ground Vehicle.

Acknowledgements

I owe a debt of gratitude to a number of people without whom completion of my thesis would have not been possible. I would like to sincerely thank my supervisor Dr. Thomas Trappenberg for his continued support and guidance throughout my research. His inspiration and prompt suggestions has guided me through my thesis. I thank everyone at HAL Lab for their support. I have learned a great deal from all the wonderful discussions we have had that helped me bring forth valuable ideas for completing this thesis.

Finally, I express my special thanks to my family, my loving parents and my wonderful sister for their unwavering support towards my goals. They have helped me through, with their encouragement.

Chapter 1

Introduction

1.1 Motivation

We use our eyes to acquire the 2D images of the world around us and pass the data on to our brain to synthesize, interpret and perceive the scene, that is to visually sense or see. Analogous to that a robot discerns the scene around it through one or more cameras and employs a set of techniques on a high powered computer to synthesize and interpret the acquired information. The science of studying and developing such techniques is called computer vision. The progress made in the last few decades has expanded its application in numerous fields like medical imaging, face recognition, biometrics and recently in transportation engaging autonomous vehicles. Computer vision extensively overlaps with two other fields, image processing and pattern recognition. Image processing is the discipline that studies operations on digital images like its transformation, restoration, enhancement etc. While pattern recognition is the discipline that studies mathematical techniques to perform operations like identifying interesting patterns in the input data, classifying input data based on their pattern and so on. Most often image processing techniques are used to process images acquired from the camera before being fed as input to a computer vision algorithm. Based on the application this could involve operations like compressing each video frame to allow computations at real-time, enhancing or softening frames of an old video to provide better input data. A pattern recognition tool can be applied to perform necessary classification of input images to supply more accurate data or simply to scale down the number of images based on a threshold.

Visual object tracking is one such research topic engaging all three fields and is indispensable when dealing with robotics. The accessibility of less-expensive high quality cameras along with the high performance processors has drawn researchers in creating a robust tracking algorithms. Object tracking as defined by Yilmaz et al. [45] is the process of estimating the trajectory of a target object in the image plane

as it moves around a scene. In general it estimates the state of the target object in each consecutive frames of a video. Such information of a target in the real world can aid a robot to move and perform actions autonomously. In essence object tracking unlocks a wide set of autonomous robot application.

Recently in robotics there has been an remarkable growth in the application of UAV(unmanned aerial vehicles) due to its commercial popularity among both military and civilians and the broad endorsement by organizations for its potential to produce technological impact. Although UAVs were primarily used by the military for reconnaissance, mapping of an uncharted territory, wild fire detection [36] [8]. Its entry into civilian domain saw application for agricultural imaging, traffic monitoring, search and rescue and more. It is important to note that the UAVs get equipped with special sensors and instruments for specific application and they are often remote controlled. This instills a range, only within which the UAV can operate. Along with dearth of UAV applications that are autonomous. It is in the interest of this thesis to present an autonomous real-time application for unmanned ground and aerial vehicles to perform long-term tracking of an object of interest in its field of view and following it. Applications for an algorithm that is fast, robust and capable of performing at real-time can be applied to HRI(Human Robot Interactions), automated surveillance, robot navigation, and autonomous aerial filming.

The work done in this thesis focuses on a long-term object tracking. That is, a tracker that works indefinitely. For this the tracker needs to update itself to recognize changes in the appearance of the target without accumulating significant error. Too much error could cause the tracker to drift away from the target. TLD(Tracking-Learning-Detection) is one such long-term tracking algorithm developed by Zdenek et al. [18]. We build over TLD making it more suitable to work on a quadcopter performing real time tracking and following with obstacle avoidance. The underlying task here is to track the object in each of the video frame and control the quadcopters actions based on the state of the tracked object. Creating such an algorithm presents several challenges [1.2.1]. This thesis is a step directed towards addressing these challenges.

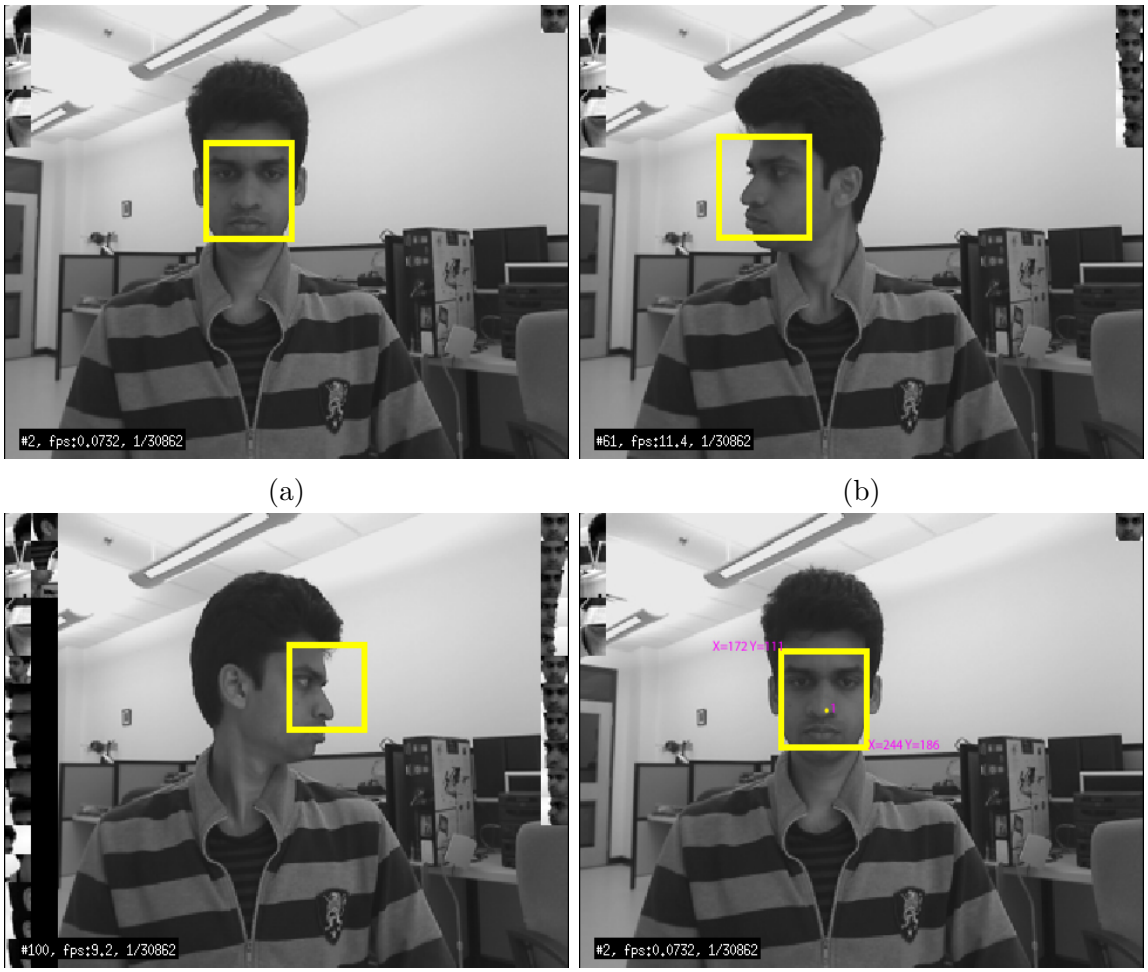


Figure 1.1: TLD long-term tracking. The target object is represented by the bounding box. Learning the changes in appearances of the target object to facilitate long-term tracking. Target object initialized in 1.1a. Representation of the target illustrated in 1.1d.

1.2 Thesis Overview

Long-term object tracking is still quite challenging even with decades of research done in this topic. In this section we shall first look at various challenges that make long-term tracking such an arduous task. Then we see why it is important to tackle these challenges by stating some of its wonderful applications in various fields of modern technology.

1.2.1 Challenges

An impacting concern with long-term tracking is caused by **change in appearance** of the object being tracked. There could be numerous activities that can contribute to this as the video progresses. The movement of the target, change in the illumination of the targets environment, change in the shape or the pose of the target are some of the most frequent. The task only gets harder when the camera source itself is in constant motion causing the viewpoint of the camera to change at every interval, which would be the case with a quadcopter. Hence these events together contribute to constant change in the view and appearance of the object to the tracking system. Now, a tracking system that has a single template as the representation model of the target object will not be able to continue tracking once there is a certain degree of change in the appearance of the target. The template will no longer be a representational model of the target. With that said, this problem does have an obvious solution.

The trackers representational model of the target should be updated by adding new templates of the target's changing appearances, allowing the tracker to adapt for long-term tracking. Inherently, this is not as straightforward. To determine how much or what portion of the new appearance is still the representational model of the original target is difficult to tell. Adapting to every change in appearance will also accumulate errors as time progresses and at one point the representational model of the tracker might not represent the target object anymore. This phenomenon is called **drift**.

Drift is handled by incorporating a learning system that can maintain a trade-off between preserving stable representation of the original target and learning the new changes in the targets appearances. Ergo a learning system is crucial for long-term tracking. The work done by Zdenek et al.[18] focuses on this aspect, to include a learning system which can perform a set of corrections on itself to reduce errors. The process and implementation of his system will be elucidated in chapter 3.

Another primary issues with object tracking is caused due to clutter in the video feed. **Clutter** is when there are multiple objects in the video that have similar features, making it difficult to discriminate the target and other objects. An example of clutter is tracking a specific car in a busy parking lot. This requires the tracking algorithm to be precise enough to not confuse the features of the target car with

features obtained from other similar looking cars.

Object trackers should also be able to handle **occlusions**. That is the target object might escape or disappear from the camera view temporarily. During this time the tracker should be able to recognize and indicate the disappearance of the target. The target may then reappear at a different location in the camera view. In this situation the tracker should be able to detect and track the object once again. In order to do this the system should incorporate a detector. The detector component essentially scans the whole frame to locate the set of features that defines the target object. Once found the detection is successful and the tracking component can actively track the target object.

Noise in the input image can also cause the tracker to accumulate errors and eventually fail. Noise could be caused due to compression, motion blur or the video itself could be degraded as a result of a faulty camera.

A tracking algorithm should tackle all the above stated challenges and still be highly efficient. Its efficiency determines its usefulness in a **real-time application**. A real-time application can involve fast moving targets, like tracking a speeding car during a car chase or tracking fast movements made by a dancer. An efficient algorithm delivers higher frames per second, which is crucial to keep up with the such fast moving target in its video feed.

1.2.2 Applications

Long-term object tracking is a very constructive tool that can be applied in various fields. Its real-time applications can reduce manual activity and motivate development towards autonomy. Some of the innovative fields in which object tracking can be applied are discussed as follows.

Augmented and virtual reality are exciting new technologies that rely strongly on object tracking to provide the precision they need to project objects onto targets in the scene or to manipulate the virtual environment based on the users eye movements for a comfortable experience. Also the ability to initialize long term tracking on-the-go on an object in an augmented or virtual world is of great interest.

In **HCI**(Human Computer Interaction) object tracking can be used to provide

interactions with computers using pure body gestures. With long term object tracking users can each generate a unique gesture convenient to them by simply initializing and learning their movements during runtime. The system can then be programmed to execute a set of commands each time the gesture is made by the user.

Another interesting application is in the field of **autonomous robotics**. Enabling **navigation** of a robot based on the object it is tracking. The constant movement of the target can be used as visual feedback for the robots movements and enable it to follow the target. Such a feature on an aerial robot would facilitate capturing photos and videos of the target object. This unbounds **security surveillance** from a stationary camera on the ground to something mobile allowing authorities to monitor and evaluate city traffic and accidents or even gauge a disaster.

Commercially an autonomous aerial robot with long term tracking capabilities can be used for **recreational purposes** too. A sports enthusiasts can initialize himself as the target and have the robot follow autonomously while capturing stills as he/she executes their stunts. This allows the user to focus on their activity and not worry about capturing videos.

1.3 Contributions

The general objective of this thesis is to construct and apply a tracking-following algorithm for UAVs and UGVs in real-time with collision avoidance. The tracking algorithm utilized here is based on the Tracking-Learning-Detection approach of Zdenek et al. [18]. Contributions of this thesis include:

- Modification of the tracking component in TLD to incorporate more robust SIFT features for refined initialization of the tracker. The scenario consists of, a long term tracker with no prior training examples and a target object initialized during runtime, both the source camera integrated on a quadcopter and the target object is in constant motion. In this regard, it is imperative to obtain prime keypoints with robust features of the target from the initializing frame for the algorithm to work.

- Reconstruction of TLD algorithm to work with the ROS framework in a Linux

environment. The ROS framework is needed to facilitates swift and easy message passing between different programs. ROS also simplifies the process of acquiring and transmitting the sensor data from the robot to the host system for processing. Then relaying back to the robot a corresponding action to execute. This form of hardware abstractions makes it simpler with minimal changes to apply the algorithm on different robots.

- Real-world implementation and validation of the proposed algorithm using the commercial AR.Drone 2.0. In this demonstration of real-time autonomous tracking and following the quadcopter has a monocular front facing camera. Ergo no depth. To maintain a distance from the target so as to avoid collision. The bounding box generated by TLD is used to map the approximate distance to the target.

- Simulation of a custom world with target object and obstacles in Gazebo. The simulation introduces a model of commercial quadcopter called Spiri. This model incorporates an on-board stereo camera. The simulated environment is used to demonstrate and evaluate our algorithm. Spiri autonomously tracks and follows the moving target Husky an Unmanned Ground Vehicle that is remote controlled.

- Incorporation of depth information to enable the algorithm to respond promptly and smoothly to sharp movements made by target object. The Modified TLD algorithm with SIFT features is supplemented with Depth information. This remarkably improves the following mechanism. The depth map is generated from the disparity of the stereo camera. The depth map also helps maintain a steady distance from the target to avoid collision.

- Performance comparisons and analysis of the obtained results. The results from the modified TLD with SIFT features is compared with traditional TLD to show the improvements during initialization. The smooth and prompt response of the quadcopter with the inclusion of depth map against the depth approximated with the traditional bounding box is also illustrated.

1.4 Document Outline

The organization of this document is as follows:

- **Chapter 2** gives an overview of the necessary background theory required for the work done in this thesis. First, the work related to tracking, detection and learning is reviewed. Next a detailed description of how SIFT algorithm generates robust keypoints is presented. Then an overview of the two unmanned aerial vehicles that are used in the demonstration along with their specification. Followed by an overview of the ROS framework and the Gazebo simulation environment used for testing and evaluation of the algorithm is given.

- **Chapter 3** presents an in-depth implementation details of each of the methods used in the algorithm. First, a comprehensive detail on the working of TLD, its structure and components. Then we have details on TLD tracker modification to incorporate SIFT and depth as features to TLD. Followed by details on reconstruction of TLD to work with ROS and notes on communication between the UAV and the host system. Next, the simulation setup, demonstration and the evaluation process is discussed extensively. The real-time demonstration setup is also similar and discussed here. Finally, details on how the depth map is generated and used in TLD for tracking and following with collision avoidance.

- **Chapter 4** describes the conducted experiments and discusses the results and observations made by providing illustrations.

- **Chapter 5** Concludes this thesis by stating the challenges that were addressed and the ones that persists. Followed by a summary on potential future research in this topic.

Chapter 2

Background

A tracking-following paradigm relies heavily on a robust long-term tracking algorithm for the feedback required to autonomously maneuver an unmanned robot to follow the target. It's a recursive process. This chapter reviews all the methods pertinent to understanding and developing such a paradigm. Sections 2.1, 2.2 and 2.3 revisits relevant approaches in tracking, detection and learning. Three essential constituents of a long-term tracking algorithm. Section 2.4 reflects on the process of feature extraction and detection suitable for objects in motion. Section 2.5 presents details on unmanned vehicles used and their hardware specifications. Section 2.6 explains the ROS framework and how the communication between the robot and the host is performed followed by illustrations of the simulation environment used for demonstrating and the evaluating the algorithm.

2.1 Tracking

Tracking defined in general terms is the process of estimating the state of the target object in each consecutive frame of a video. Given a live feed or a video sequence having frames $I_1....I_n$ the objective is to estimate in each frame I_k the state of the target object X_k. The state of the target can be a combination of variables containing data on object shape, location, color, pose and scale. Combining two of more of such characteristics uniquely identifies the object in the scene. Once defined the state of the target can be represented by set of points, a bounding box, elliptical patch or even as a silhouette or contour of the object as shown in figure 2.1 [45]. A tracker that represents its target object by a point is called a point tracker. They are best suited for objects that don't change their shape or scale and are small in size. Multiple points, contours and silhouettes are best suited to represent a non-rigid body like that of an animal. Geometric shapes like a rectangular patch or an elliptical patch are suitable for target objects that experience considerable amount of scale change

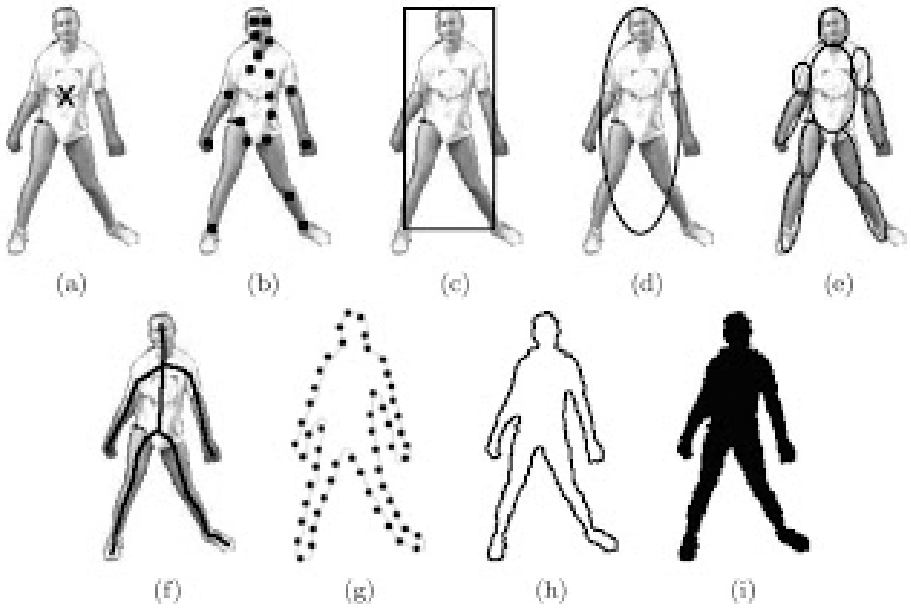


Figure 2.1: Common object representations [45]. (a) Centroid (b) Multiple points (c) Rectangular patch (d) Elliptical patch (e) Part-based multiple patches (f) Object skeleton (g) Control points on object contour (h) Complete object contour (i) Object silhouette

and in-plane rotations. The part-based multiple patches as shown in figure 2.1(e) are useful when the target is a composed of several rigid parts that are constrained by certain geometric relation.

The rectangular patch will be used in our implementation to keep up with the scale changes, in-scale rotations and also to give a better estimation of the objects motion. The rectangular patch will be referred to as the bounding box as it encapsulates the target object. In figure 2.2 below, the bounding box defines the target in the current frame I_k and its state X_k is represented by the top-left and the bottom-right pixel coordinates of the bounding box.

Tracking algorithms can be manual, automated or semi-automated. A manual tracking requires human interaction to define the target in each frame. Automated tracking initializes the tracking process automatically in the presence of the target, but needs to have a priori information about the target to work. Then there is the semi-automated method. This requires a human interaction at the beginning to select the target object in order to initialize the tracking process. A semi-automated paradigm is advantageous when the target has to be selected during runtime with

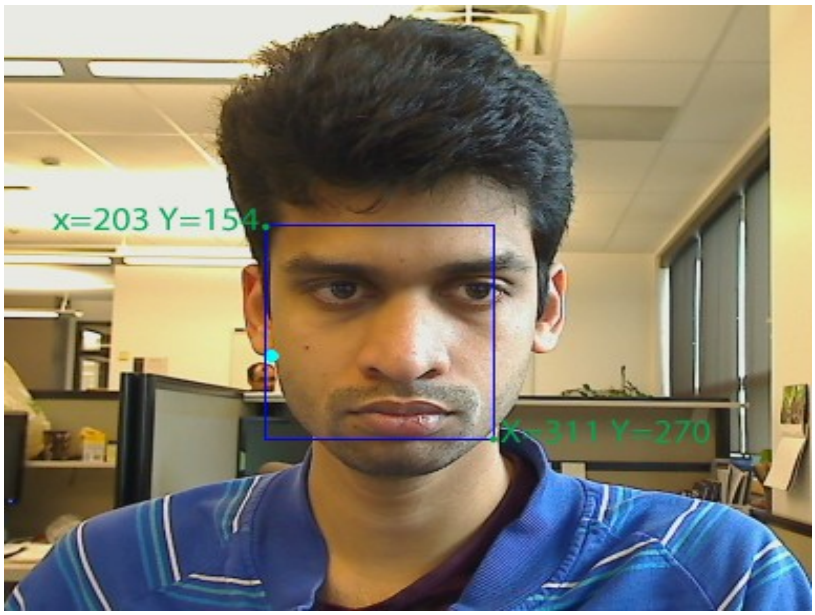


Figure 2.2: Lucas-Kanade feature tracker [1] [23]. The target object being the face, is represented by a bounding box. The illustrated X and Y values mark the top-left and the bottom-right pixel coordinates of the bounding box.

no prior knowledge. Having a target selected, the next step is to represent it in a bounding box and track it through consecutive frames. This brings us to the different tracking methods that can be applied depending on the application domain.

2.1.1 Tracking Methods

Object tracking methods are classified under three principal categories. They include Point tracking, Kernel tracking and Silhouette tracking [45].

- In **Point Tracking** the target object is represented by a set of points and their association between the current and the previous frame is based on their position and motion.
- In **Kernel tracking** the appearance of the target object is represented by a geometric shape such as an ellipse or a rectangular template. These templates, containing the model of the target object are searched in each consecutive frame to obtain a match. Matching in successive frames formulates the needed motion estimation.
- The **Silhouette tracking** methods rely on encoding shape model of the target

object. The shape model can be in the form of color histogram, edges or object contour. The tracking in successive frames are done by matching shapes and contour evolution of the target object.

We will only examine the concepts of point and kernel tracking methods as they are pertinent to our implementation. Lucas-Kanade is a popular point tracking method, that works based on three assumptions to track similar points from one frame to the next [3].

- **Brightness consistency**: The first assumption states that pixels of the target object will not change its brightness as it moves from one frame to the next. Expressed in equation 2.1. The pixel at two dimensional image coordinate $X = (x, y)$, while changing its location from one frame to the next will still retain its brightness value.

$$I(X_1) = I(X_2) \tag{2.1}$$

  Where I represents the intensity value. X_1 and X_2 represent pixel location in video frame 1 and 2 respectively. $X_2 = X_1 + d$ with d referring to displacement vector.

- **Temporal Persistence**: The second assumption states that, the displacement made by the pixels from one frame to the next is small. That is the motion of the target object is slow relative to temporal increments. Equation 2.2 illustrates that d is small.

$$X_2 \approx X_1 + d \tag{2.2}$$

- **Spatial Coherence**: The third assumption states that, all neighboring pixels within a small window around the target pixel moves coherently [42].

Figure 2.3 illustrates these three assumptions. These assumptions will be applied in our implementation along with other error measures to track the points representing the target object to obtain the flow from one frame to next.

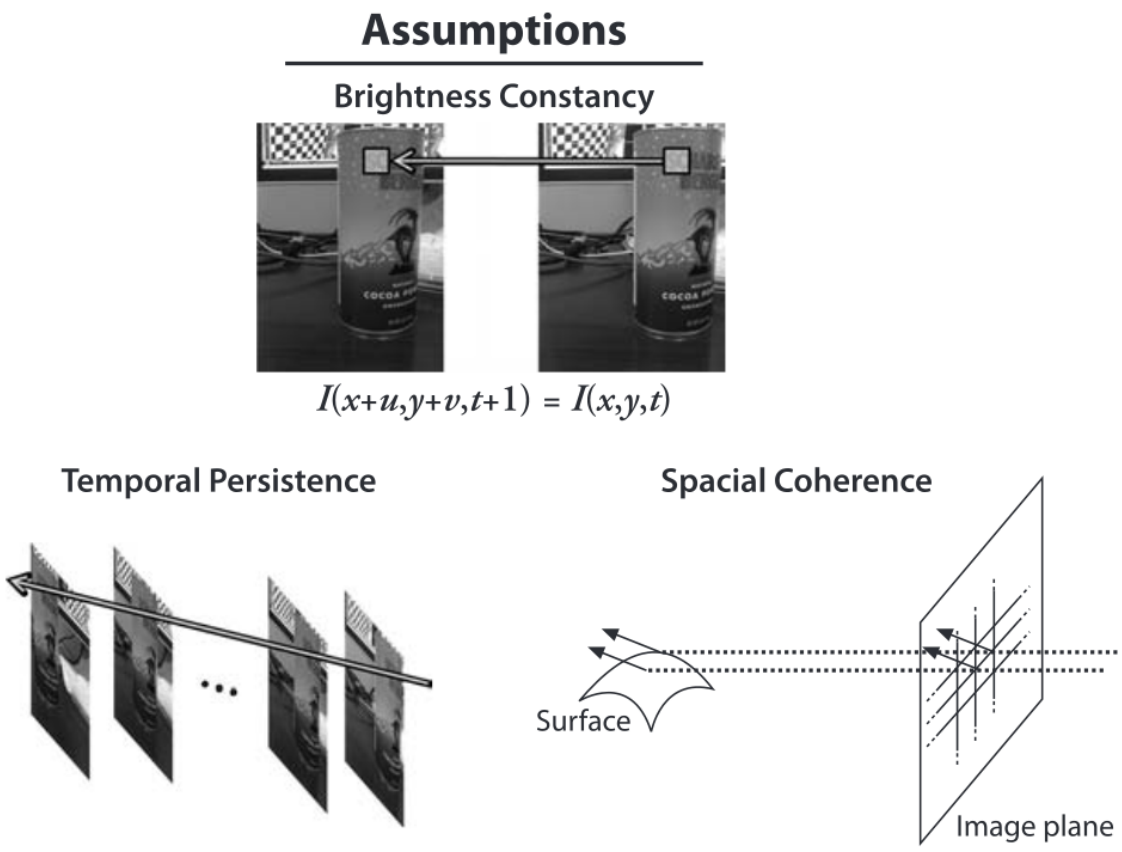


Figure 2.3: Figure illustrates the three assumption made by Lucas-Kanade. The image in the top row shows the consistency of the patch brightness given the displacement of u and v in dimensions x and y while moving frame t to $t+1$. The image for temporal persistence shows the slow movement of the target object relative to temporal iteration of video frames. The image for spatial coherence depicts, coherent movement of neighboring pixels of the target object. [3]

On the other hand, in the method of kernel tracking the objective is to model the appearance of the target object and represent it within a geometric shape. This is done by acquiring the portion of the frame that contains the appearance of the target object. This portion is called an image patch or a template. Once acquired the tracker then performs a sweeping search using a window of size equal to the image patch through the subsequent video frame to find a similar template based on a predefined similarity measure. This method of scanning the frame exhaustively from top to bottom is called the sliding-window technique. Each patch under the sliding-window can also be scaled by preset steps to improve robustness. The template with

the highest similarity would be the target object and the displacement from the previous frame will be then determined. Tracker of this kind is called a template tracker [26].

Performing an exhaustive template matching over the entire frame is time consuming and inefficient [4]. Hence, in order to increase run-time efficiency the newer algorithms for template matching limit their search around the vicinity of the pixel position from the previous template. This is based on the reason that, given the template and its position in the previous frame it is safe to say that the matching template in the current frame would still be in the vicinity of the previously known position, provided the object hasn't made an improbably large displacement or been occluded. This concept allows the tracker to restrict its search area. Thereby greatly improving its efficiency [33]. Gradient based optic flow method is one such technique that performs template matching based on the previous pixel location. The gradient ascent/descent based maximization/minimization method easily provides the transformation model to predict the movement direction of the target object for the next frame. Figure 2.4 shows the result of such method performed by Jepson et al [14].

Mean-shift algorithm [6] is another popular technique used for performing template matching around a given pixel location. In contrast to the gradient based technique a mean-shift tracker characterizes the target by a color histogram. The tracker then compares the histogram with the surrounding windows to maximize the similarity measure. This is done iteratively until convergence. The similarity measure is based in terms of Bhattacharya coefficients [7] as illustrated here,

$$\sum_{u=1}^{b} P(u)Q(u) \tag{2.3}$$

where P is the color histogram of the target template, u is the corresponding feature vector, Q is the templates around the target pixel location from the previous frame. b is the number of bins in the color histogram.

Collins et al. [5] then proposed a technique to further improve the efficiency of a tracker and reduce the number of false matches. The idea is to discriminate the target object from its environment. By removing the background we have fewer templates to cover during matching and as a result fewer chances of false matches. This

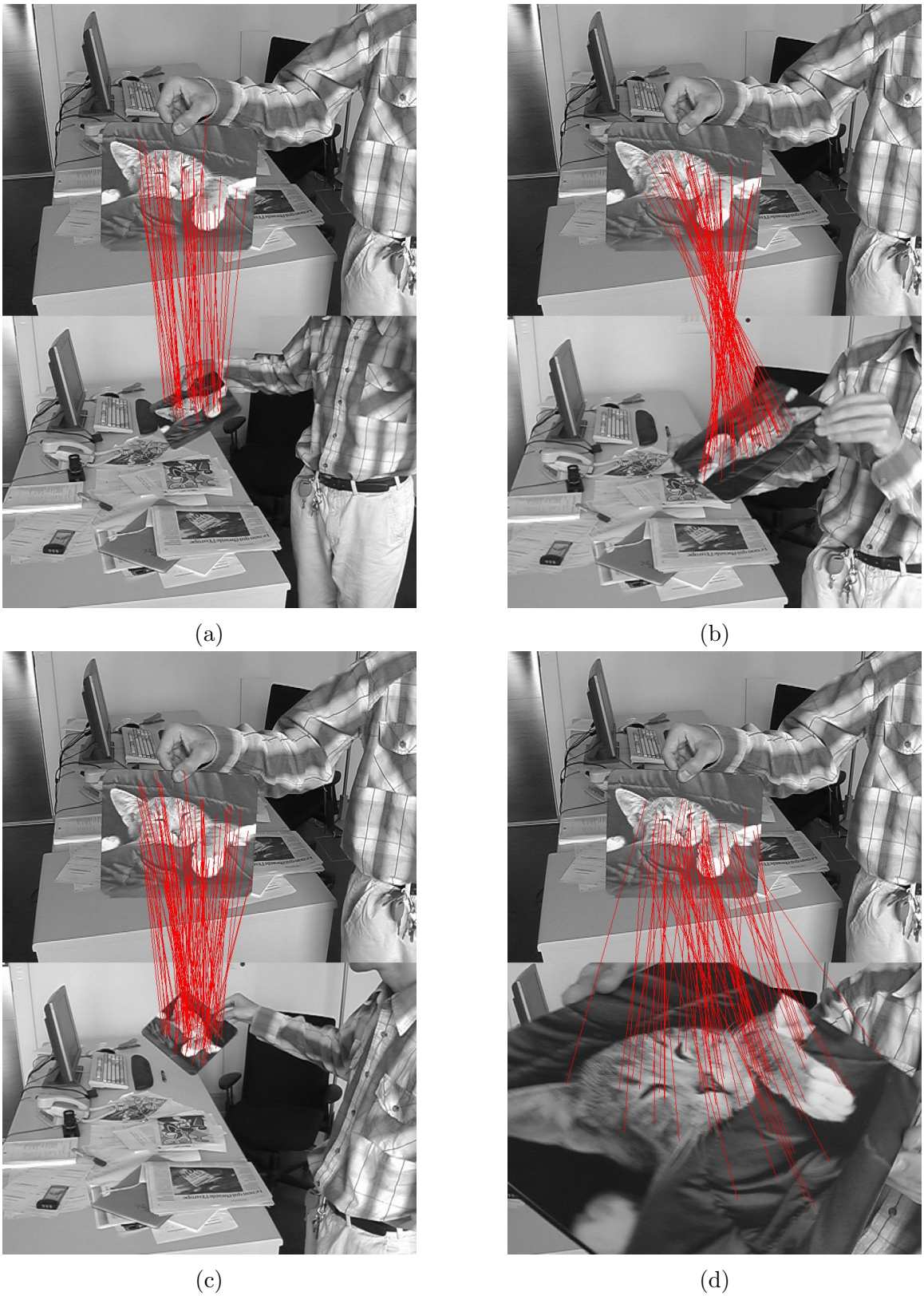
Figure 2.4: Robust online tracking by Jepson et al. [14]. (a) Target object inside an elliptical kernel in successive frames. (b) Stable components obtained from the target patch that are reliable for tracking by applying gradient based technique

is done by binary classifiers to distinguish the object of interest from its background. Collins et al. performed classification based on discriminative color space of the image.

Tracking by itself is only effective as long as the target object remains within the field of view. If the target object is occluded or if it makes a rapid movement fast enough to slip through to a farther pixel location, the tracking process would halt, indicating no target found. This is more likely to happen as the target moves in the real world interacting with other objects in the scene. To obviate the halt, a detector system needs to be initiated, that can scan through the current frame to locate the target object and dispense the position to the tracker to resume tracking.

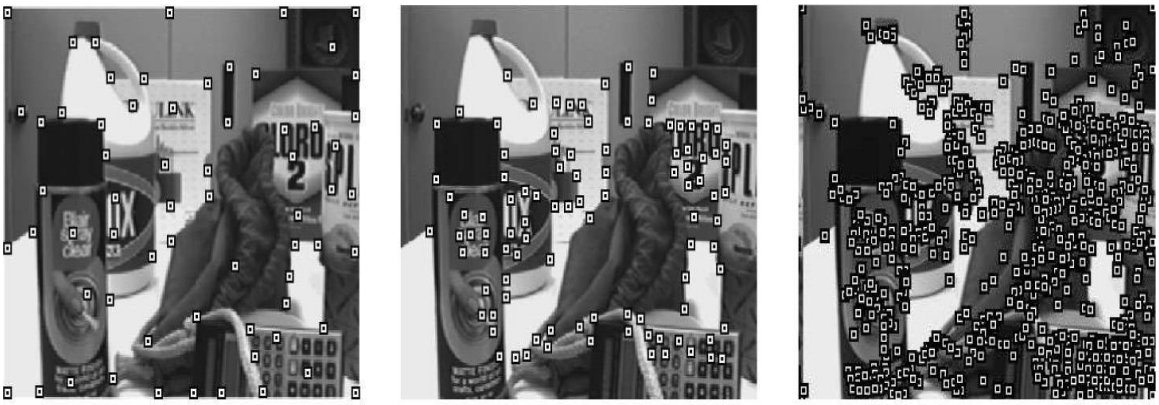
2.2 Detection

Object detection is an essential mechanism that is either invoked on each frame by the tracker or only during its initialization. An object detector's chief function is to identify each of the various objects in the current frame, determine the target object among them and update the tracker with its position. This is done by clustering the pixels of each of these objects. In most cases, the object of interest being tracked is a known visual class. Hence it is feasible to train the object detector beforehand allowing it to detect the target among various other objects in the video frame. It should also be noted that a pre-trained detector by itself can replace a tracking mechanism completely by simply applying detection in each frame, provided there isn't much change to targets appearance. Such methods are referred to as Tracking-by-Detection, Ozuysal et al. [28] developed a detector that applies warping techniques to generate multiple warped templates of the target object. It further trains the detector on these templates. The off-line trained detector then performs pairwise pixel comparisons to find the target object in each frame as shown in figure 2.5. Though this detector is efficient at detecting the target object. It can't handle change in object appearance and also the absence of on-line training makes it inadequate for several real-world applications. What we require is a detector that can be trained on-line by acquiring more representations of the target object during run-time while complementing the tracking process to reduce errors.



(a) (b) (c) (d)

Figure 2.5: Ozuysal et al. [28], detection-by-tracking done by performing pairwise pixel comparison. The features are generated using SIFT. Detects target object with changes in scale and rotation. But, with no change to targets appearance



(a) (b) (c)

Figure 2.6: Interesting points detected by (a)KLT (b)Harris and (c)SIFT feature detectors [45]

2.2.1 Detection Methods

We shall further go through various prevailing methods applied for object detection that are pertinent to our work.

Point detectors

Detecting interesting points in an images is the most elemental form of object detection. Each object in the images would have its local texture. Points from these localities, that are invariant to a combination of scale, illumination, rotation and viewpoint are said to be interesting. In an image the term point refers to the smallest unit of information, a pixel. There are several models that work based on point detection. Like the Harris interest point detectors [12] , Kanade-Lucas-Tomsai detector [35] and SIFT detector [22]. We use SIFT feature detector in our implementation, section 2.4 gives in-depth details of its process of feature extraction. Harris point detector and the KLT detector focuses on intensity variations in the image. In essence theses detectors detect corners and edges in the image. Harris detector first measures the x and y directional intensity variations by applying first order image derivatives in x and y directions, (I_x, I_y) of the image. Then encodes the aforementioned variation measures into a second moment matrix $(\boldsymbol{M})$ and this matrix is evaluated for each pixel of a small area in the image. Matrix M is as illustrated in equation 2.4 [12].

$$M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \tag{2.4}$$

Next the detector computes determinant and the trace of M. It is as illustrated in Equation 2.5. Here k is a constant and the measured R represents the Interesting point confidence. In order to select the interesting points the detector performs thresholding on the measure of R for each of the small areas in the image.

$$R = det(M) - k \times tr(M)^2 \tag{2.5}$$

The results obtained by this procedure is shown in Figure 2.6(b). The KLT detector follows similar steps as well, except that it gauges minimum eigenvalues of M to compute the interesting point confidence, R and then thresholding is done to select the points. This helps KLT detector discard points that are too close to one another in each of the small areas of the image. The interesting points detected by KLT is shown in Figure 2.6(a). It can be seen from the results that the only compelling difference is the spatial distance kept between the interesting points in the KLT detector.

Both KLT and Harris detectors are invariant to rotation and translation of the objects in the image but they are still bounded by affine transformations. This brings us to the more resilient SIFT detector. It generates interesting points that are invariant to rotation, translation and affine transformations. It also generates more flexible keypoints than the other detectors due to its process of pyramidal construction of an image to different scales and acquiring interesting points from each of them. Each process stage of the SIFT detector is elucidated in Section 2.4 to assist with our implementation .

Background Subtraction

Another intriguing way to identify objects in an image is to distinguish them from the background. Given the model of the background to train the detector, it can apply background subtraction methods to glean divergent object regions in the image. The conventional approach [13] is to locate regions that deviate significantly

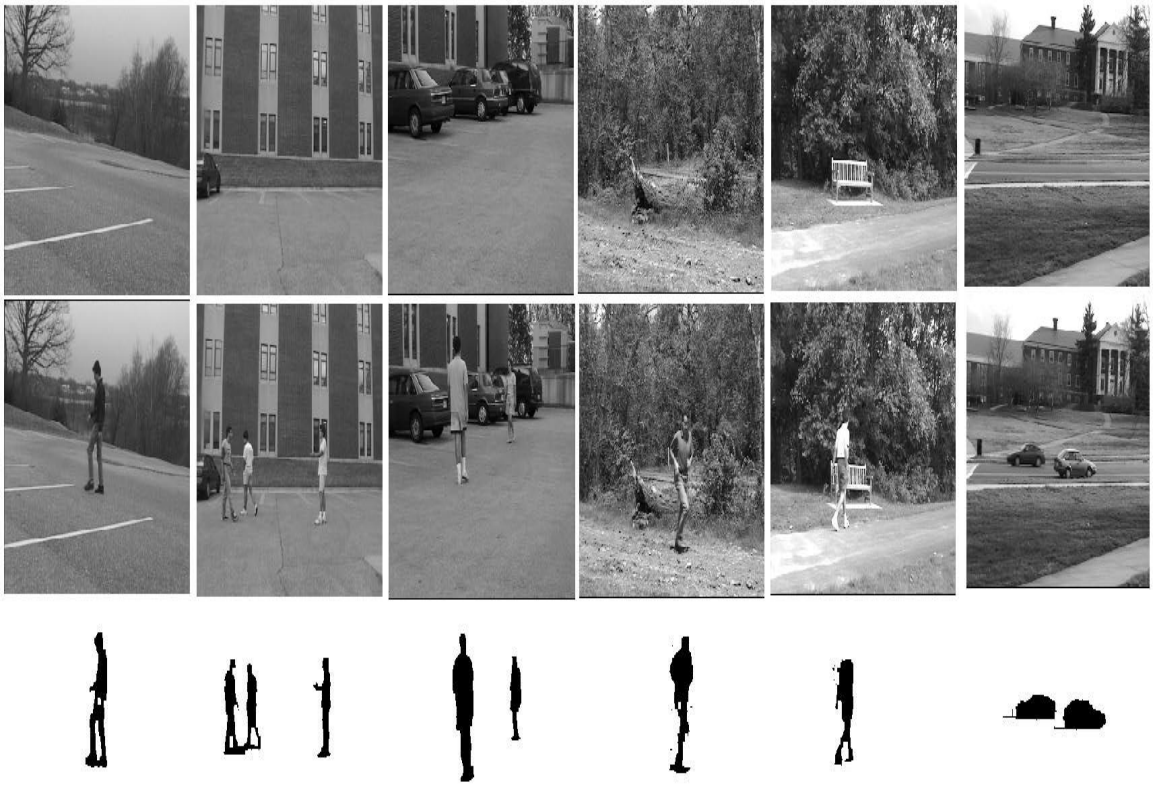


Figure 2.7: Background Subtraction, work done by Haritaoglu et al. [11]. The first row exhibits a set of background scene models. (NOTE: The scene also includes regions that are not completely stationary like, swaying trees). The second row exhibits the foreground objects in the scene. Third row highlights the constructed appearance of the foreground objects in the scene.

from the background model indicating a moving foreground object. Each of the pixel from the deviating region is separated from the current image to construct the object appearance, hence it is called background subtraction.

The immediately apparent flaw with such an approach is dealing with a background scene containing non-stationary regions like swaying trees or changing illumination. Figure 2.7 shows the work of Haritaoglu et al. [11]. This algorithm is able to deal with non-stationary background regions, noise and changing illumination. This is accomplished by applying a multimodal model to depict the background scene colors [10] [40]. That is each pixel of the background model is subjected to a mixture of Gaussian distributions [41] to generate more variations of the background scene color per-pixel. During detection, each pixel of the current image is compared with

each Gaussian variation of the background model until a match is found. For each pixel the mean of the highest weighted Gaussian represent the persistent color and the lower weighed Gaussian represent the less frequent colors of that pixel. Thus, if a pixel in the current image match for the lower weighed Gaussian will be identified as a foreground object.

The background subtraction approach is most suited for fixed cameras performing motion detection and surveillance and not suitable for a camera that is moving constantly generating new viewpoints. But the underlying idea of separating the background from the object of interest is compelling. The TLD algorithm applied in our implementation stores the background templates from each image to provide the detector with negative examples for its on-line training. Detecting and discarding the background from the current image drastically improves the efficiency of the algorithm. As stated in section 2.1 Collins et al. [5] based their work on background subtraction to treat detection as an aspect of classification between the target object and the background. In order for the detector to automatically learn the diverse views of both the background and the target object, it requires a learning mechanism. In the next section we shall look at some of the machine learning methods applied to object detection.

2.3 Learning

The two common scenarios with object detection is either having a large collection of manually labeled examples or having a very limited amount of labeled examples. In the latter case pre-training an object detector is not an option. A **semi-supervised learning** system needs to be applied to automatically learn different views of the object being tracked along with the background scenes. It is called semi-supervised as it acquires new unlabeled training data during runtime and labels them based on its limited set of manually labeled examples. When performing such a classification the emphasis should be on learning the right representation model of the target object and to avoid accumulation of errors. In essence the core of the detection process is to perform binary classification on small patches of the current image and classify them as either the target object (positive examples) or the background (negative examples).

(a) (b)

Figure 2.8: Supervised learning applied to pedestrian classification [44] (a) Positive training examples of pedestrians. (b) Marked boxes show the detected pedestrians in each image

In this section we shall look at some of the supervised and semi-supervised machine learning methods applied to perform such classification.

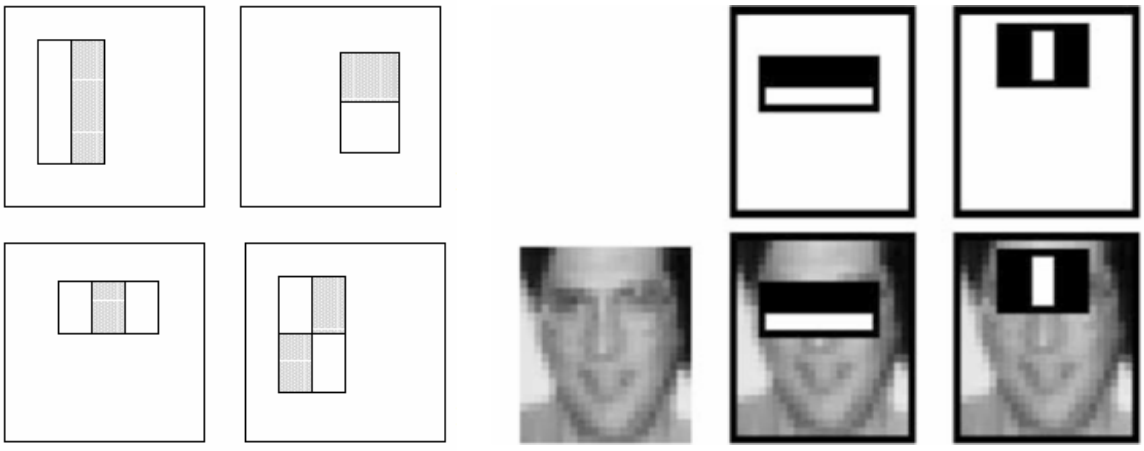
2.3.1 Supervised Learning

In applications where the class of objects to be trained on is already known with a large collection of manually labeled examples available. It's conventional to apply supervised learning method to train the detector. A supervised learning system is one which, based on prior labeled examples (x_i, y_i), where $x_i \in X$ represents set of training examples and $y_i \in Y$ represents set of class labels, defines a function $f : X \rightarrow Y$ that maps input data to the desired class. In the case of object detection, having two object classes manually defined. Each of the sample patches are characterized by points in a feature space. The objective is to determine a decision boundary to discriminate features of one class from another. That is separating positive examples from the negative. For instance, detection of pedestrians based on their motion appearances is performed using a supervised learning methods [44] as shown in Figure 2.8. We shall discuss one such popular supervised learning methods that is relevant to our work.

Adaptive Boosting

Boosting is an iterative process of improving a classifier's performance to provide a more accurate classification, hence the term boosting. It was introduced by Freund and Schapire [9]. The concept is to combine multiple base classifiers that are not quite accurate themselves, but can work together to yield a better classification. The initial step of this algorithm is to construct a distribution of weights over the set of training examples. These weights represent the focus of difficult examples and are set to an equal value at first. After the first iteration of classification, the system selects the base classifier that has the least error. All its associated misclassified data are given higher weight value. The increase in weights emphasize their difficulty. In the next iteration the remaining set of base classifiers are trained to classify the more difficult data set and so on. This is iterated until each of the base classifiers have learned to classify progressively complex features. Finally the weighted combination of all these base classifiers gives us a more accurate overall classification.

A classic application of boosting is the adaptive boosting algorithm developed by Viola et al.[43] in 2003 for their Viola-Jones face detection. The base classifiers used are simple perceptrons trained on image features. The features from the image



(a) (b)

Figure 2.9: Adaptive Boosting [43]: (a) Four forms of rectangular haar-like filters. The difference between the sum of pixel intensities in the white region to the dark region gives the feature value of the region. (b) First row shows the two filters selected by adaptive boosting mechanism. Second row illustrates that, the eye region is darker than the cheeks. the eyes are of darker shade than the nose

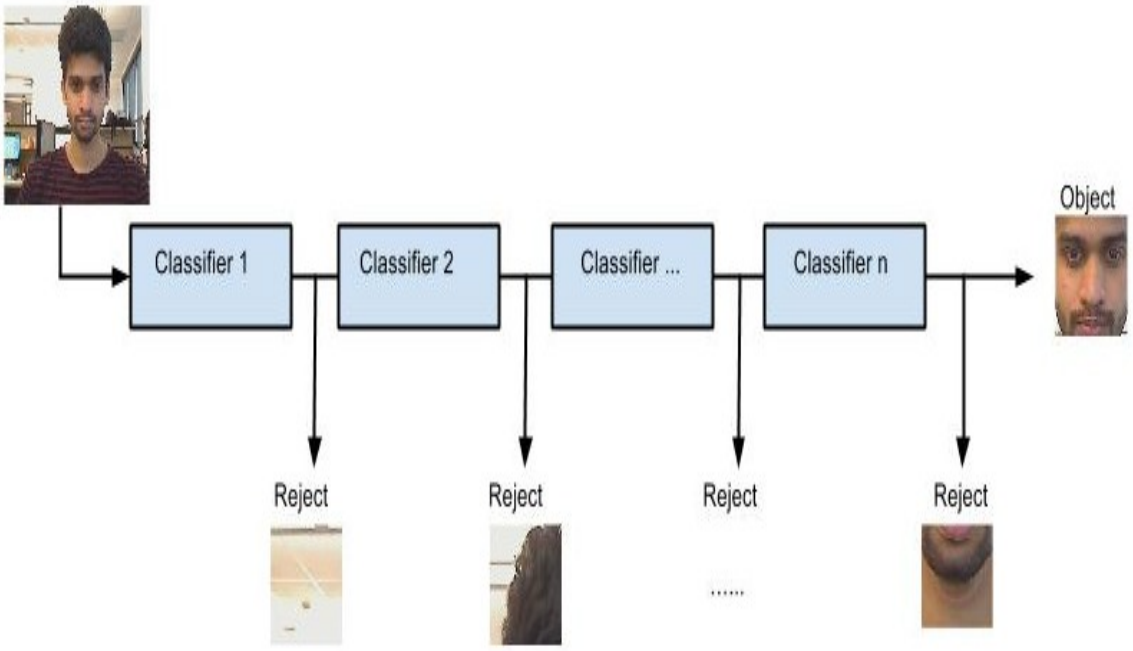


Figure 2.10: Cascaded adaptive boosting: Each stage has a single base classier performing classification on the input image that is split into thousands of smaller patches. The negative examples at each stage are dropped. This approach curtails false positives and reduces computational cost

are extracted using a combination of rectangular filters as shown in figure 2.9a. The feature of a region in the image is the difference between the sum of the pixel intensities in the two shades on the filter, shown in figure 2.9b.

Viola et al. improvised the standard boosting mechanism to perform cascaded classification. For this the input image was divided into thousands of smaller patches and presented to the system as the input. Where at each stage a single base classifier performs classification on all the image patches and the resulting negative examples are cut short from advancing to the next stage. This way each progressive stage has fewer number patches to classify and as a result greatly reducing the computational cost as well as reducing the number of false positives. Figure 2.10 illustrates the cascaded adaptive boosting process. It can be seen that the major portion of the image contains the background. Discarding these regions at an early stage is essential to make the system perform at speeds suitable for real-time applications.

Figure 2.11: The object motion forms a trajectory in a video stream. This spatial and temporal dependency of any image patch provides some structural constraints that aid the semi-supervised learning system in classifying unlabeled data

2.3.2 Semi-supervised Learning

In an application where the target object for tracking is selected on a single video frame during runtime, only one positive example of the target object with a small set of negative examples acquired from the region around the target is available. The objective of the learning system in this scenario is to learn the classifier a function that can classify all the unlabeled data in the consequent frames based on labeled examples from the target initializing frame. Then apply the newly labeled data to retrain the classifiers to learn more features and further improve the detectors classification. This is an iterative process with more freshly labeled data added to the increasing training set of examples with each subsequent video frame. Such a learning system built on both labeled examples and unlabeled data is termed as semi-supervised. We shall further review a semi-supervised learning paradigm termed PN-learning developed by Kalal et al. [15] that is employed in TLD.

PN-Learning

The notion behind semi-supervised PN-learning is to make good use of the abundant unlabeled data available with each consecutive video frame during object tracking. In a more general scenario the unlabeled data would usually be unrestrained by any dependency making it difficult to perform reliable classification [2]. But, with

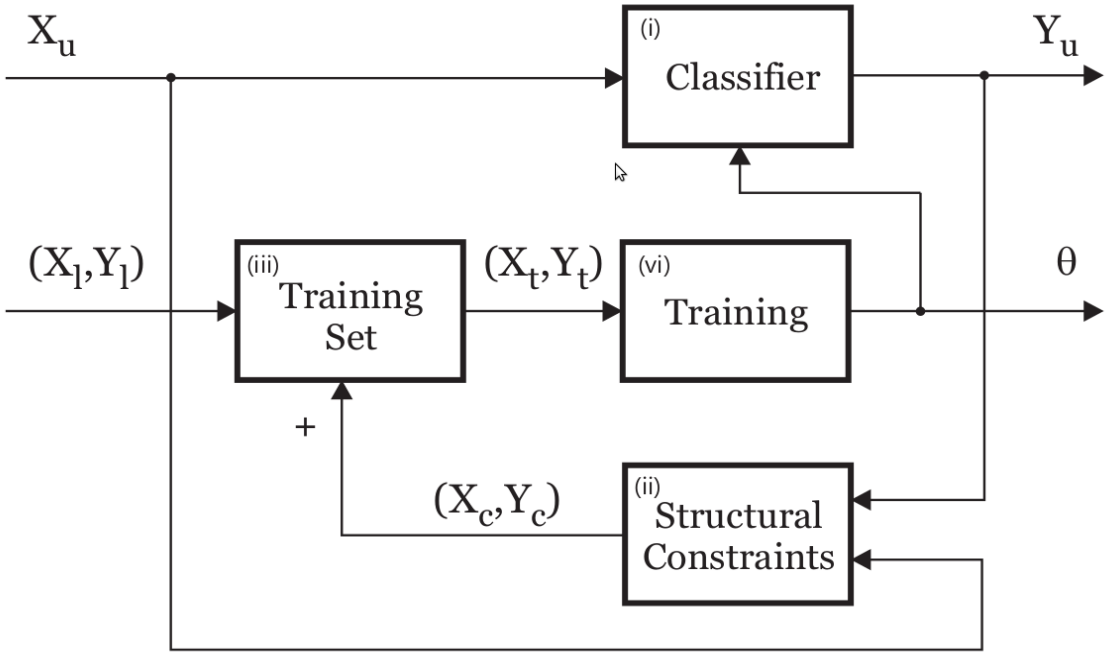


Figure 2.12: Block diagram of P-N learning mechanism [15]

respect to object tracking there are always spatial and temporal dependencies [15] making it possible for the learning systems to exploit the unlabeled data reliably. The dependency is that, in each frame of a video, the unique target object can only occupy one patch in the image and the object forms a trajectory in the successive frames. It is as illustrated in the Figure 2.11. Hence its possible to define a structure based on this object trajectory. The patches that highly overlap the target object is classified as positive examples while the patches that are farther away constitutes the background, classified as negative examples. As the system tracks the object patch it learns new appearances of the object along with more examples of the background.

What makes PN-learning unique is the enforcement of these two particular constraints on the unlabeled data classified by the detector. The system can analyze and re-label the errors made during classification based on the aforementioned spatial and temporal structure. This decreases the errors in the training set and improves the quality of classification with each iteration. We shall discuss these qualities of PN-learning in detail in the remainder of this section.

Figure 2.12 [18], depicts the components that constitute the PN-learning mechanism. It contains a binary classifier, set of labeled examples called the training

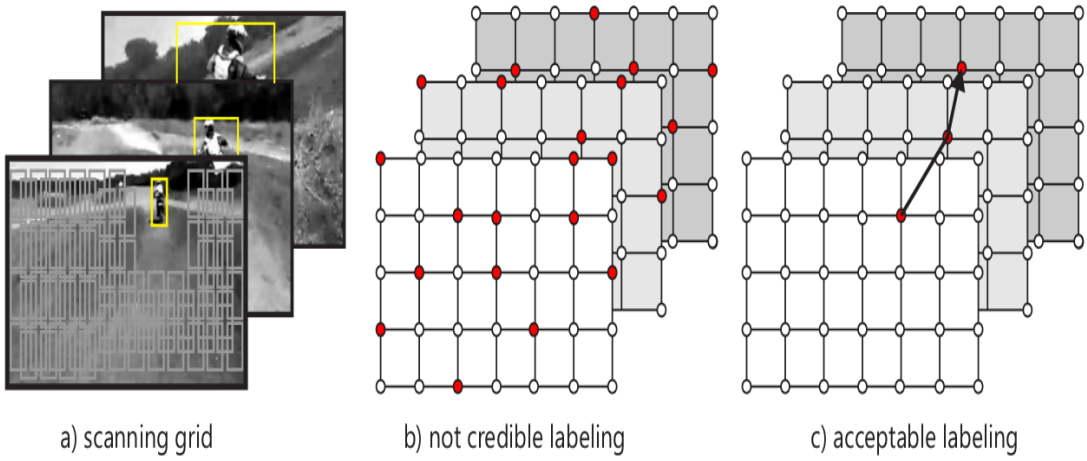


Figure 2.13: (a)Scanning grid spans the whole image to felicitate evaluation of smaller individual patches by the binary classifier (b) The bold dots indicate the multiple patches classified as positive example. This violates the spatial constraints (c) Expected adherence of classified positive examples to the temporal structure defined by the target object through consecutive frames [18]

set, a supervised learning method applied to train the classifier on the labeled data in each iteration and a component that analyzes the classified data to enforce the spatio-temporal constraints by relabeling errors made by the classifier.

Now given a video stream for object tracking, we have two labels, positive and negative labels in our label space Y and a single patch of the target object selected in the initial frame as a positive example x_1 from the feature space X. The object detector then scans the whole image by applying the same technique as Viola et al. [43] discussed earlier. The image frame is divided into sub images or patches of size similar to the bounding box of the target object selected in the initial frame. For a typical image frame of size 240×320 pixels, there would be about 10^5 patches to be evaluated by the classifier. In the initial frame, the patches that considerably overlap the target patch are labeled as a positive examples and the remaining non overlapping patches that comprise the background are labeled as the negative examples. These labeled examples together form the **training set** L.

A supervised learning method applies the labeled training set L to learn the **binary classifier** a function $f : X \rightarrow Y$ parameterized by Θ. Once trained the task of the binary classifier is to evaluate each of the **unlabeled data** X_u in the next

image frame and to classify them. To do this the detector slides horizontally from top to bottom spanning the whole frame. The binary classifier on each patch decides whether the patch contains the target object or the background and labels them appropriately.

The classified data is then evaluated based on the two constraints, P-constraint and the N-constraint. With the location of the current object patch and its trajectory defined by the tracker. The **P-constraint** assesses each of the negative examples to see if they adhere to the temporal structure defined by the tracker. A patch that follows this trajectory in consecutive frames of the video has to represent the object. If a negative example overlaps this trajectory it is regarded as a misclassification or a false negative and the patch is relabeled as a positive example.

Algorithm 1: PN-Learing Mechanism

Input:1)Labeled example from previous frame: Object patch(O)
2) Unlabeled data acquired from current frame

Output: Labeled example set

for $t = 1 \rightarrow \infty$ **do**
  Train classifier on labeled data
  Classify unlabeled data $X_u \rightarrow Y_u$
  `// N-expert`
  for $Positive(Y_i) = 1 \rightarrow n$ **do**
    Estimate false positives
      $Negative(Y) \leftarrow Falsepositive(Y_i)$
  end
  `// P-expert`
  for $Negative(Y_i) = 1 \rightarrow n$ **do**
    Estimate false negatives
      $Positive(Y) \leftarrow Falsenegative(Y_i)$
  end
  Retrain the classifier with updated training set
end

The **N-constraint** similarly checks to see if each of the classified positive examples adhere to the spatial structure defined by the tracker. In any given frame the object can only be present at a single location. If multiple Positive examples not overlapping with the previously known object location are found, they will be regarded as false positives and relabeled as negative examples. Figures 2.13b and c visualize the labelling constraints. Following that, these enforced examples are used as the training set to train the binary classifier for the next iteration. With each successive iteration the training examples are corrected and the binary classifier is trained to avoid repeating previous errors. Pseudo-code for the for PN-learning is as shown in Algorithm 1.

Detection, tracking and learning works based on features extracted from the image patch containing the target object. In the next section, we shall look at how these important and robust features are extracted to facilitate long-term tracking.

2.4 Feature detection

In order to perform effective tracking and detection, the algorithm needs to be able to locate the object of interest in each of the captured frame. This calls for careful feature detection among different images or consecutive frames of a camera feed. As with long-term tracking the object-of-interest isn't necessarily at rest. Ergo the feature detection algorithm needs to be robust enough to identify similar features even though the object in the image might be subject to change in rotation, scale and translation.

One such popular algorithm in computer vision is SIFT. It stands for Scale Invariant Feature Transform. It was developed by David Lowe [22]. SIFT processes each image through several stages to obtain distinctive keypoints each with robust features. The keypoints from the consecutive images are matched by a detection technique to find the object of interest. This gives us the flow of the object or indicates occlusion. This section further explains the set of steps performed by SIFT to obtain scale, rotation and translation invariant keypoints.

First stage This stage of the algorithm prepares the image to make it scale invariant. In order to do so, it creates a scale space of the original image. That is,

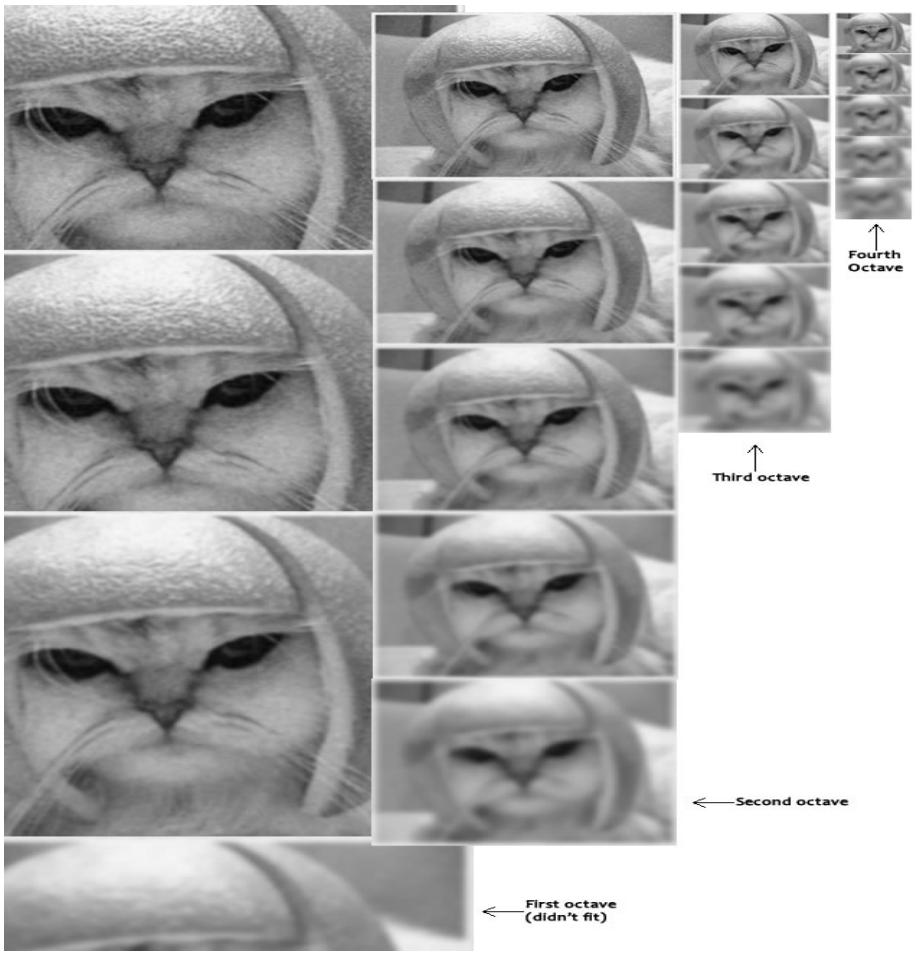


Figure 2.14: SIFT Scale Space [37]. Images of the same size represents an octave. Here there are 5 images in an octave, each formed by introducing progressive amounts of Gaussian blur

generates multiple images from the original image that are progressively blurred out. This blurring is done by adding Gaussian blur. It then reduces the image size by half and generates more images with progressive blur and so on. This process is repeated a set number of times.

Based on the application we can make the decision on how many levels of scale are required. As shown in the Figure 2.14 the vertically stacked images form what is called an octave. Each octave has images of the same size but with progressively higher scale(gaussian blur). The number of octaves generated can be varied and so can the number of scales in each octave. The scaling or blurring effect is the convolution of the gaussian operator and the image pixel. Equation 2.6 [22] illustrates the convolution operation.

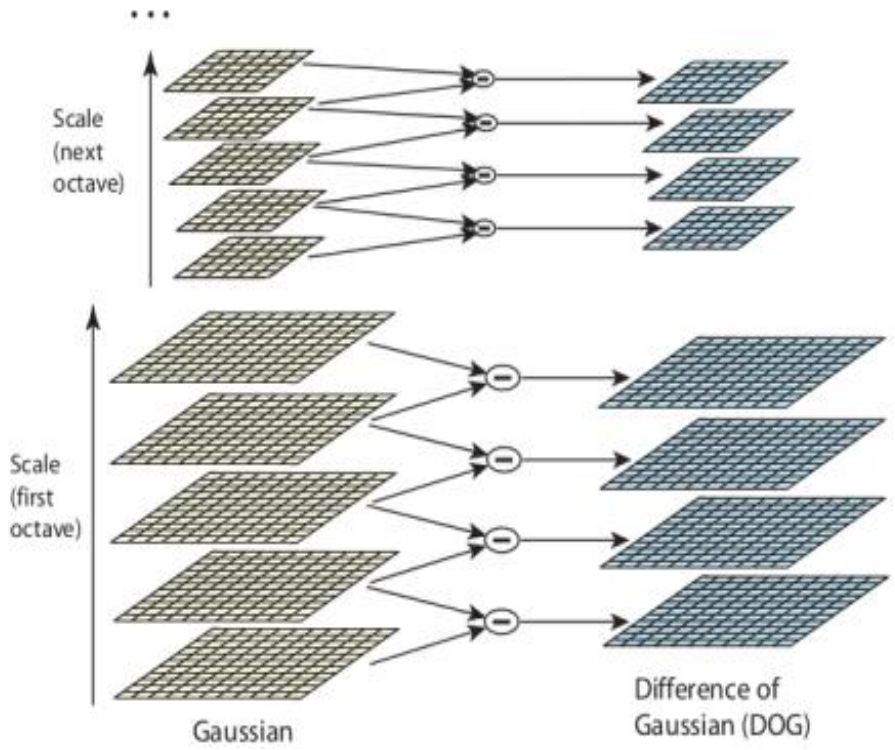


Figure 2.15: Consecutive images in scale space are subtracted with each other to from the Difference of Gaussian. Processes is repeated for all the octaves. The DOG images generated are also scale invariant and approximate to the Laplace of Gaussian [22]

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{2.6}$$

Here L represents the resulting blurred image, G is the Gaussian blur operator and I is an image. The x,y indicates the pixel in the image coordinate to which the operator is applied, while σ specifies the amount of blur. Higher the σ, higher is the effect of blur.

Second stage In this the objective is to detect corners in the image. Locating them is essential for generating interesting keypoints. Basically, this can be achieved by calculating second order derivatives of an image and its scaled counterpart. This would have to be repeated for each consecutive scales in all the octaves. Calculating second order derivative for all those pairs is computationally intensive and the resulting images would also be highly sensitive to scale [37]. Hence a work around as

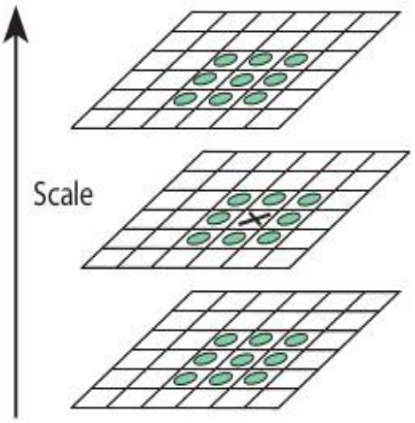


Figure 2.16: Locating maxima/minima. X indicates the keypoint under inspection. The circles represent the neighboring points. Minimum of three scales are considered with 26 neighbors to compare with for each keypoint

suggested in [22] is to calculate the difference between two consecutive scales in the scale space. The process is as illustrated in the figure 2.15 The resulting image is called the difference of Gaussian which is approximately equal to the Laplacian of Gaussian. By performing this, we not only negate computationally intensive operation but also generate images that are scale invariant.

Third stage Now that we have the difference of Gaussian images, we move on to finding some interesting keypoints. The first step of the process is to find the rough location of maxima or minima in the difference of Gaussian images. The pixels obtained by this step is still only an approximate of the maxima or minima. In order to obtain the exact points we have to go between pixels hence it needs to be located mathematically.

To illustrate, figure 2.16 shows the first step of identifying approximate locations of the maxima and minima. For checking a pixel in the current image, the two closest images are considered. One image of higher scale and one of lower scale. The pixel under check marked X in the figure will be checked with 26 of its neighbors. It can be immediately skipped if it fails the maxima/minima check with any one of its 26 neighbors. For all the maxima/minima pixels identified, the sub-pixel values have to be calculated to further narrow down on the exact maxima/minima. This is calculated by performing Taylor expansion of the image around each of the marked pixels. These

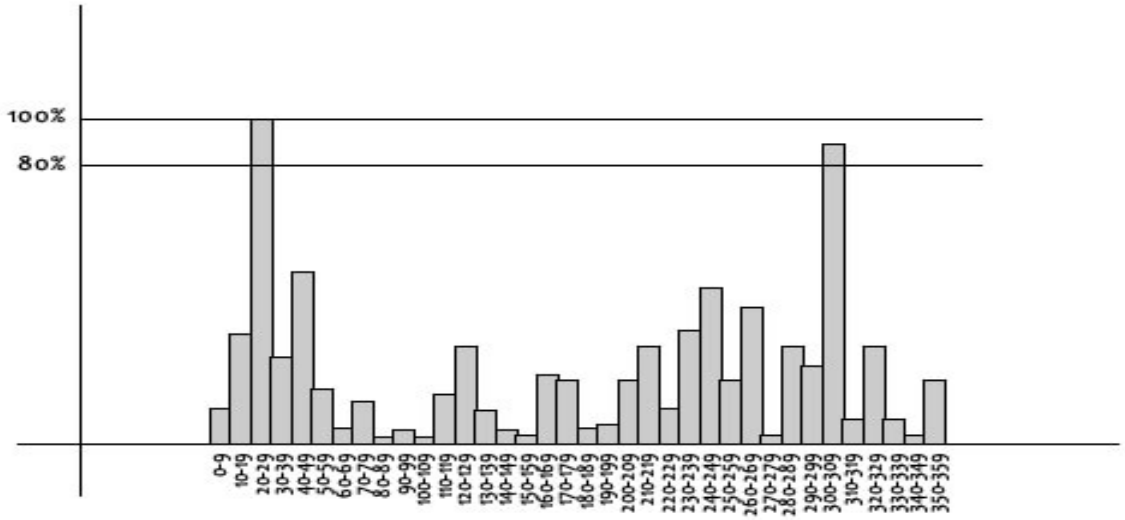


Figure 2.17: Keypoint orientation histogram. 36 bins spanning, 0-360 degrees. The orientation of the keypoint is assigned based on the histogram peaks [37].

sub-pixel points are imperative to the robustness of this feature matching algorithm.

Although this step generates some very robust keypoints it also produces a lot of keypoints that are not any good as a feature. Hence, It is considered best to remove them at this stage. The SIFT algorithm discards features having low contrast in its intensities. As the keypoints check for maxima/minima they also check for the intensity of the current pixel in the difference of Gaussian image. If it is less than a certain set threshold it will be discarded.

Along with low intensity keypoints SIFT also discards the keypoints that are in flat region or part of an edge. The algorithm is more interested in points that are part of corners. SIFT uses Hessian matrix [22] to detect if a point is a corner or an edge or neither. Basically, the logic is to calculate two perpendicular gradients at the keypoint. If both Gradients return a small value the keypoint lies in a flat region and it can be discarded. If one of the gradients returns a large value the keypoint is part of an edge. If both gradients return a large value, its a corner.

Fourth stage At this point we have stable keypoints that are invariant to scale. In this stage each of the keypoints will be assigned an orientation to achieve rotation

invariance in feature detection. This will greatly increases the stability of the keypoints. To determine the orientation the magnitude and gradient direction around each keypoint is determined. With this data an histogram is created. Figure 2.17 shows the histogram. The 360° of orientation is divided among 36 bins in the histogram. In the figure the histogram peaks at bin with orientation range 20-29°. The orientation of the keypoint belongs to this bin which is the 3^{rd} bin from the left in the histogram. Hence the orientation assigned is 3. Another thing to notice in the figure is, there happens to be another bin that peaks above 80% . Any peak above 80% is identified as a new keypoint. The new keypoint would have same scale and location as the original but with a different orientation corresponding to its peak in the histogram. Now we have stable keypoints that are scale invariant and rotation invariant.

Final stage of SIFT. Now that the keypoints have both scale and rotation invariance. SIFT creates a unique fingerprint for each of the stable keypoints. The reason for creating such a fingerprint is to allow for easy identification of the keypoint during feature matching.

As shown in the figure 2.18, a 16×16 window around the keypoints is broken down to sixteen 4×4 windows. Each of these 4×4 windows has its orientation and

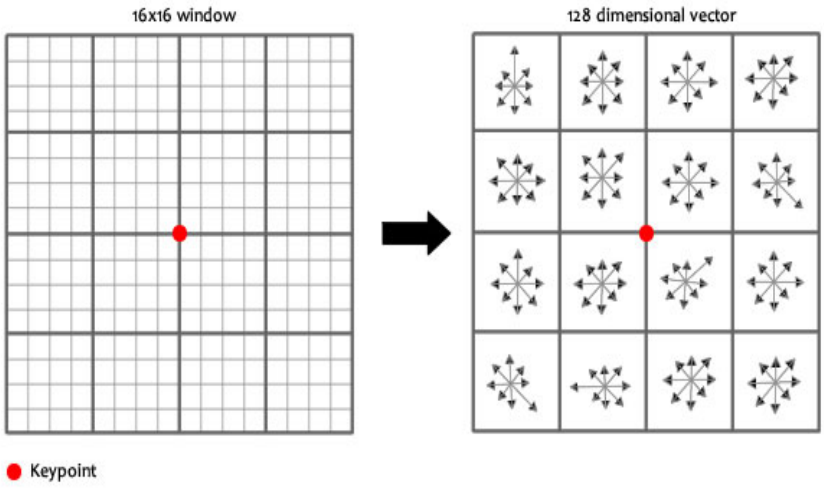


Figure 2.18: To create unique fingerprint for each keypoint. A 16×16 windows around the keypoint is considered. It is then broken down into sixteen 4×4 windows. Each window contains the gradient magnitude and its orientation [37]

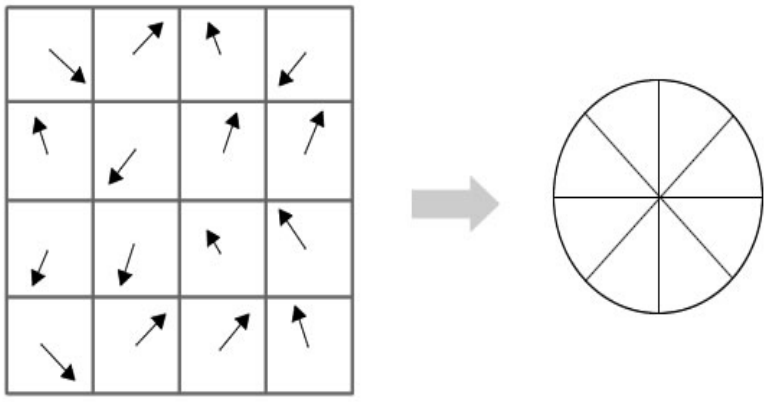


Figure 2.19: Random gradient orientation of each of the 4×4 windows is mapped to pre-defined eight ranges or orientation. [37]

gradient magnitude calculated and graphed using a 8 bin histogram. Each bin has an orientation range of 44°. So for instance, if the gradient orientation on the first windows is 63°. It would get added to the 2^{nd} bin. The amount contributed to the bin depends on both the magnitude of the gradient and its separation from the keypoint. Hence the gradients farther from the keypoints contribute relatively smaller values to the histogram. Applying this method to each of the 16 windows, converts random orientations into 8 predefined range as shown in figure 2.19 specified by the histogram bins. Since each keypoint has sixteen 4×4 windows and each of the 4×4 windows carries 8 orientation values. We get a total of $16 \times 8 = 128$ values.

These 128 values uniquely identifies this keypoint and is referred to as the feature vector of this particular keypoint. Once we obtain such feature vectors for a good number of distinctive keypoints we can apply them to plethora of applications. It is popularly used in robot localization and mapping [34], image stitching, scene modeling, recognition and tracking. We will incorporate SIFT keypoints and its feature vectors in TLD and compare the results with that of the tracking by TLD using regular grid based feature extraction in chapter 3. In the next section we discuss about the hardware that will be used for our implementation.

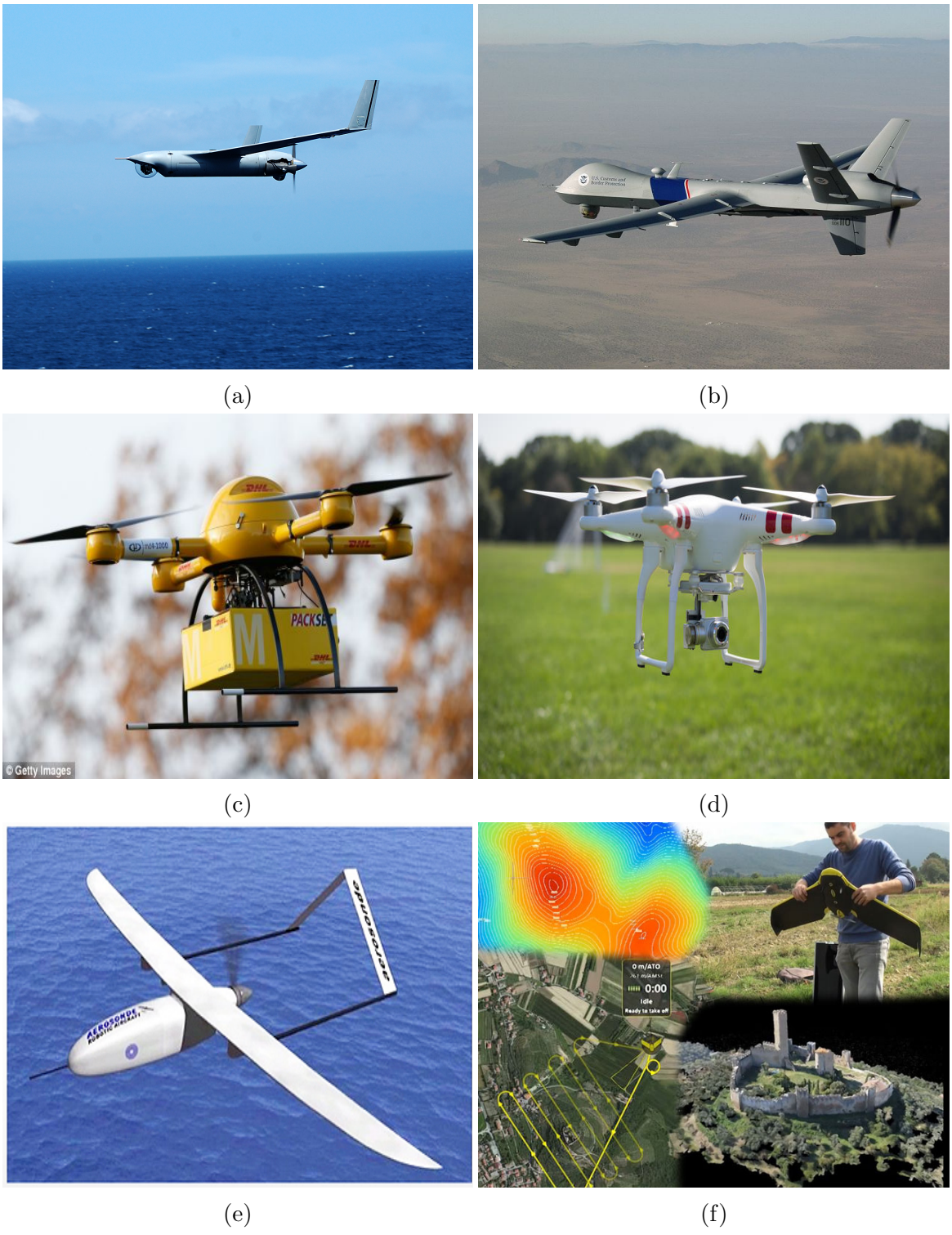


Figure 2.20: Applications of UAV: (a) ScanEagle by Insitu [29], employed for fish spotting (b) MQ-9 Reaper by General Atomic used by US Air force for border surveillance (c) DHL testing its UAS parcel delivery service (d) DJI Phantom, used for professional aerial photography (e) AAI Aerosonade by Aerosonade Ltd, designed to collect weather data (d) eBee by senseFly is equipped with instruments for surveying

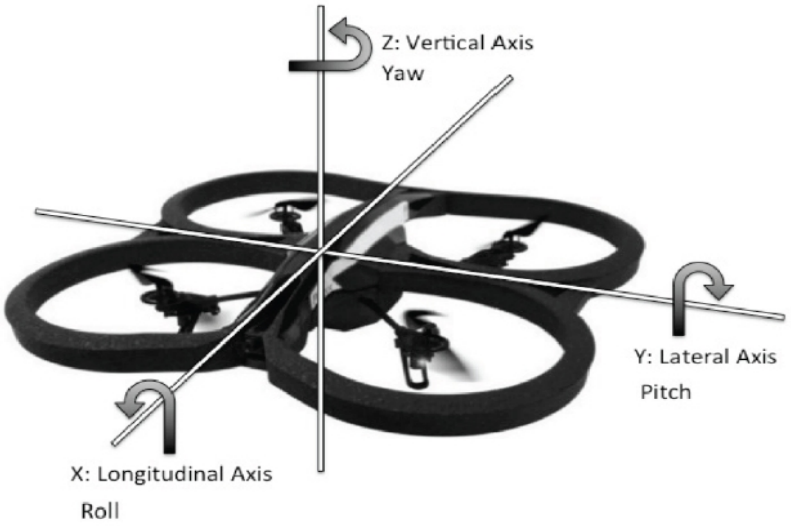


Figure 2.21: AR.Drone 2.0 [19], illustrates the three rotation dynamics (Yaw, Pitch and Roll) that define the orientation of a vehicle moving freely in a three dimensional space

2.5 Unmanned Aerial Vehicles

Quadcopters are a class of unmanned aerial vehicles (UAV). They are propelled by four rotors. Due to their multirotar dynamics, quadcopters are inherently hard to fly and handle. But, thanks to the recent technical developments, quadcopters come with automatic on-board stabilization hence allowing users to use them out-of-box without worrying about any technical aspects like, tuning the rotors for stability. This led to its exponential rise in popularity over the years and kept it in the light of research in various fields. Researchers are currently focused on making intelligent application for UAVs. These UAVs can be modified to accommodate numerous sensors making the use-case possibilities infinite. They are starting to be employed in a wide range of fields, from agriculture for chemical spraying and inspection [27] to surveillance by police. DJI as shown in Figure 2.20d for instance is one of the popular commercial quadcopter manufacturer that direct their product sales toward activities like candid filming, professional photography, obtaining thermal imaging data for research and even for crowd surveillance for large events. Figure 2.20 highlights some of the popular fields in which UAVs are an asset.

For our research we have utilized the commercially available quadcopter called AR.Drone 2.0 as seen in Figure 2.21. It has been popular among researchers in the

lab environment for its relatively low cost and its small, lightweight form making it perfect for flying in an indoor environment. They come equipped with numerous on-board sensors. For monitoring its motion, it combines a 3 axis accelerometer and a 3 axis gyro to form of a 6 DOF (degree of freedom) MEMS (micro-electromechanical system) IMU (inertial measurement unit). The IMU measurements are used for automatic stabilization of the drone's pitch, yaw and roll motions, shown in Figure 2.21. The drone for its altitude measurements relies on two on-board instruments. The ultrasonic range finder attached at the bottom of the hull provides stable measurements up to 6 meters. Above which, the drone relies on the barometer to provide it with altitude data for stability. It also has an on-board application processor that serializes all the data from these sensors facilitating easy accessibility for the user.

For our implementation we primarily rely on image sensors. AR.Drone comes equipped with two cameras, one front facing camera with 92° wide angle lens capable of capturing HD (1080*720) video at 30 fps and one bottom facing camera capturing QVGA (320*240) at 60 fps that is up-scaled to 720p while streaming. The image data from the primary front facing camera will be provided as the input to TLD. The on-board WIFI adapter on the AR.Drone allows it to stream the image data to a host system running our algorithm. Due to AR.drone's front monocular camera we work around the inability in creating a depth map by using the properties of the bounding box enclosing the target object to maintain a fixed distance to the target. We go over the details in Chapter 3. A depth map is a vector containing information relating to the distance from an object in the scene from the viewpoint. In order to be able to generate this depth map the drone needs to house a stereo camera. We were able to discover a startup company named Pleiades that was working on building exactly that, a drone with on-board stereo camera capable of capturing videos in 3D. It was targeting the 3D video market which is currently flourishing in the cinemas with all major movie's coming out in 3D and even at our homes with 3D televisions.

Pleiades is currently working on their prototype, which they named Spiri, shown in Figure 2.22. As there was no hardware to work with, they provided us with their full fledged simulation of Spiri with all the expected features in a stand-alone package called Gazebo. This simulation environment will be explained further in the following section. Spiri houses a stereoscopic front facing camera that can capture 1080p image

Figure 2.22: Spiri by Pleadies

data at 30 fps. It has an on-board processor running Linux Ubuntu 14.04 distribution. This provides the prospect of running the implementation right on the drone. Thus negating overhead caused by transmitting data back and forth to an host system at home base. The image data acquired by the left and the right front facing cameras contains disparity of the scene relative to each other. Using this we can develop the necessary depth map that can be used in our implementation to maintain essential distance from the object of interest and avoid obstacles. In the next section we provide details about the ROS framework and the Gazebo simulator.

2.6 ROS and Gazebo

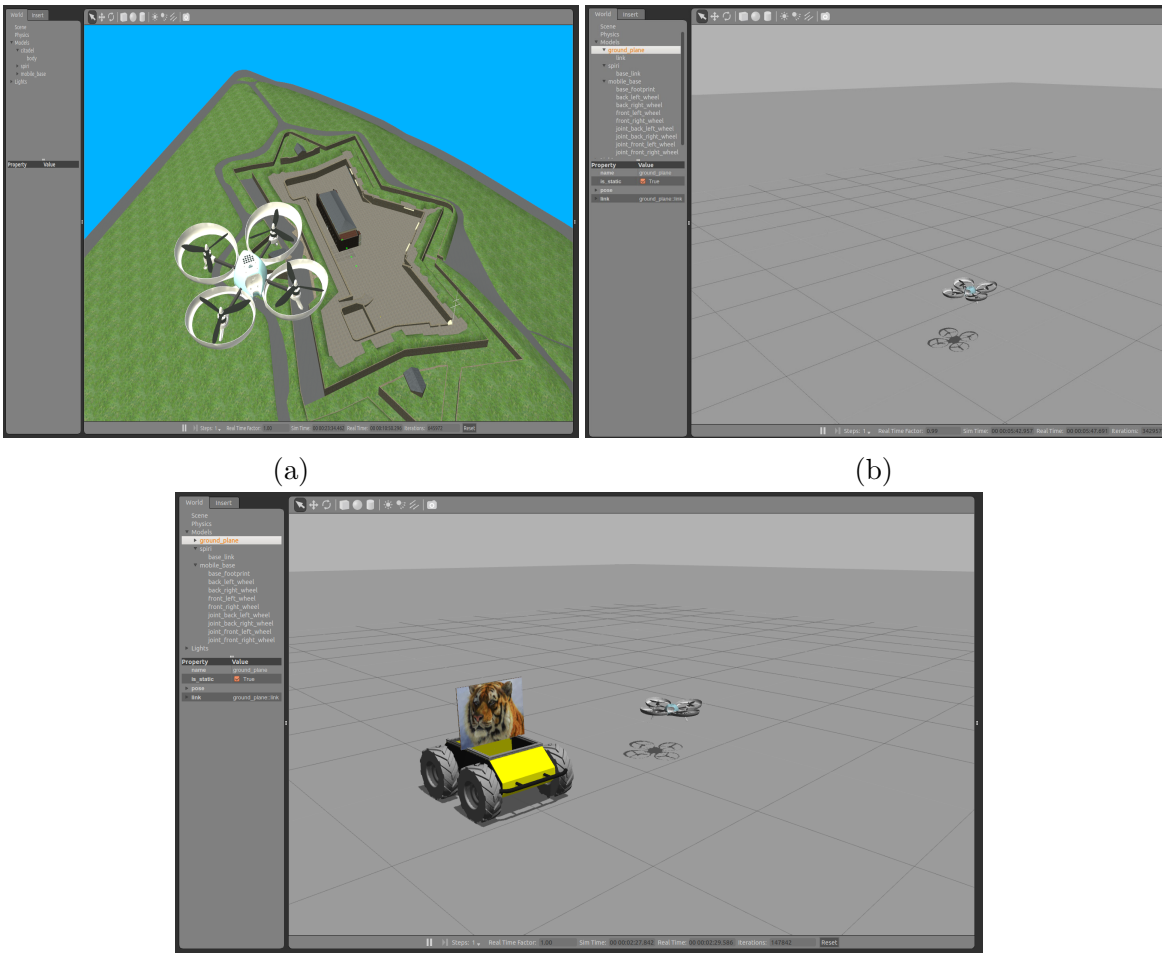
ROS is an open source Robotic Operating System. Similar to a standard OS, The ROS framework provides necessary hardware abstractions enabling users to interface with a plethora of robots being manufactured in the market. It provides communication between components and processes in the form of a publish-subscribe messaging infrastructure [31] [20]. Each program that can be run is called a node. Nodes can have several named buses called topics that can exchange messages with other topics. For instance we could have a node for a robot, whose purpose is it to present us with specific sensor reading that can be used by other nodes to perform specific tasks. This robot node would interact on the low level with the sensor hardware to acquire say,

barometer and thermostat measurements, process them to obtain the robots altitude and publish this information as a message via a topic name. Other nodes interested in the altitude information for their task would simply subscribe to the aforementioned topic and receive the message as soon as they become available. Passing data in this form allows us to naturally develop a distributed computing system.

ROS being open source, it is constantly improved and supported by a large community. Over the years it has accumulated an extensive set of tools developed by manufacturers, researchers and robot enthusiasts. There are tools to configure a robot, for debugging and logging, there are also some well defined visualization tools to process the camera feed from the robot. As most of these tools rely only on the data from the sensors and not the overall specification of the robot, they work independent of the robot in use.

In our implementation ROS allows us to communicate with the AR.Drone, access the sensor and camera data and publish them on several topics. For instance, the TLD implementation in Matlab along with numerous other nodes subscribe to these sensor topics. TLD processes the image data to perform the object tracking. It then generates the bounding box and publishes this data. The AR.Drone movement node subscribes to the TLD topic to get the location of the object in the 2D pixel coordinate. It performs the necessary mapping to the 3D world coordinates to generate the vectors for the drone's movement and then publishes it. This vector contains the needed pitch, yaw and roll angles for the drones along with the velocity by which it needs to make these angles. Finally a node containing the binding low level libraries subscribes to these vectors to pass the data to the velocity controller. There-by enabling the drone to track and follow the object of interest. This was the cursory overview of how nodes interact to perform the tracking. Next, we shall look at the simulation environment.

Gazebo is a 3D robotic simulator [25], an environment in which we can create any 3D scenario to test our algorithms or build robots from scratch, create our own models for obstacles and terrain. Some of the popular robots are pre-built and can be added to the environment immediately. Gazebo being open source and managed under BSD license is supported by a large community of researchers and enthusiasts. Similarly



(a) (b)

(c)

Figure 2.23: Gazebo environment powered by ROS (a) Spiri flying over a model of Halifax Citadel hill, provided by Pleiades (b) Spiri flying inside a map, we call Testing grounds (c) Spiri and Husky

Pleiades has also created a model of Spiri on Gazebo with all the features that will be supported by their real marketed piece. Figure 2.23b shows Spiri in the empty Gazebo environment. Figure 2.23c shows Spiri with another land based robot, the Husky. A popular favorite among robot enthusiasts and researchers. Figure 2.24 depicts a graph known as the RQT graph. It's a ROS plug-in for visualizing the computation and connection in the form of graph representations. In the figure, nodes are represented by ovals and the topics are represented by rectangles. Communication between two nodes is indicated by a line arrow connecting them through a topic. The node at the tail of the line arrow is the node publishing a topic, while the node at the head is subscribing to the topic. The interactions, i.e. the publishing and the subscriptions

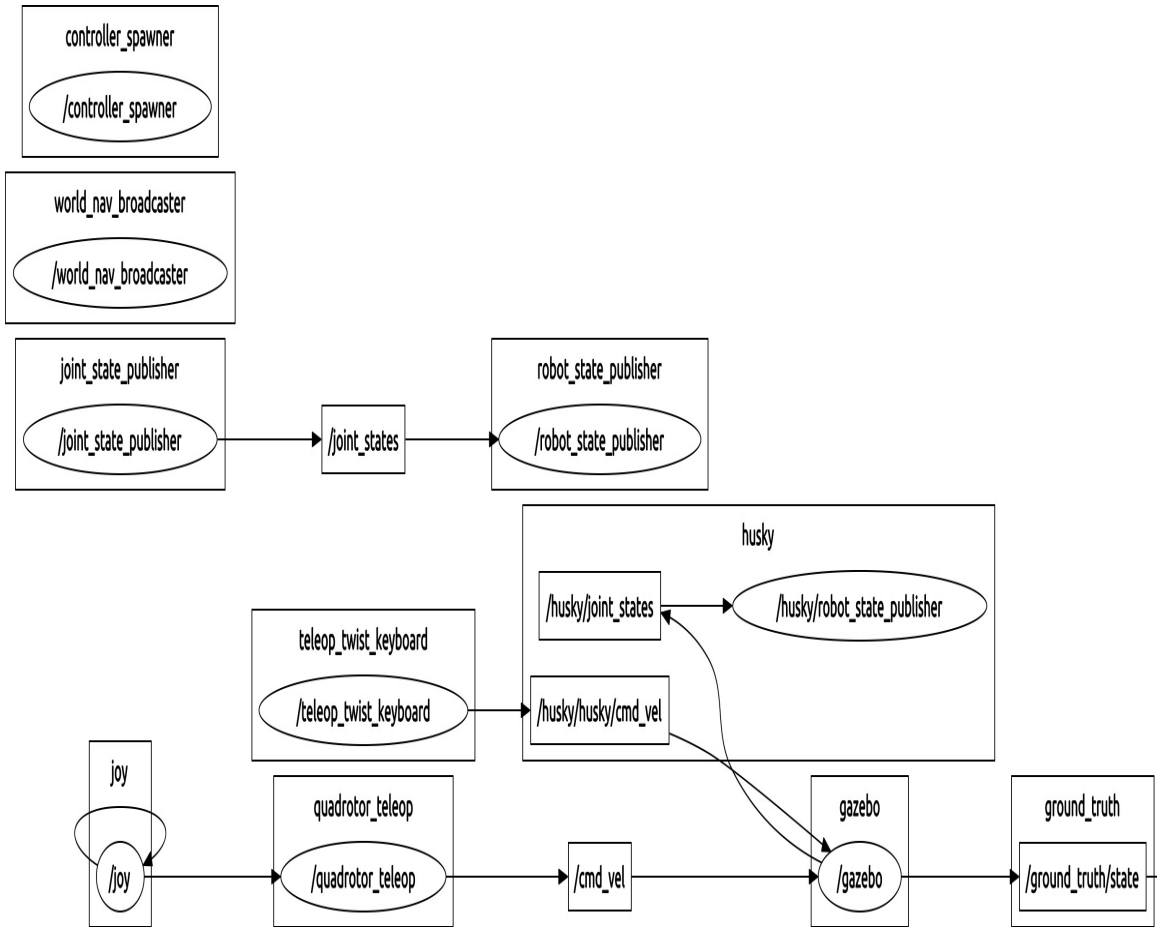


Figure 2.24: An RQT graph depicting active topics and nodes in gazebo environment holding a model of Spiri and Husky shown in Figure 2.23c. Nodes and topics are represented by ovals and rectangles respectively.

between nodes and topics can be seen in clear structure. This plug-in is part of the bigger package called the RQT. It is a QT based framework capable of implementing various GUI tools in ROS. The RQT graph of our implementation will be elucidated in Chapter 3.

Essentially once a robot is built in the gazebo environment. ROS listening to the same port as Gazebo discerns the simulated robot as the physical robot. ROS then serves as the interface for the simulated robot for us to carry out the testing. Gazebo is know for providing solid physics, object dynamics and good illumination effects in its environment. It employs multiple open source engines like Bullet and Open Dynamics Engine [38] to produce gravity, inertia, collision dynamics. Hence it's preferred to apply any prototype algorithms inside a simulation to test and observe.

Once a satisfying result is obtained it can be applied to a robot in the real world.

Spiri will be placed along with a Husky in Gazebo. The Husky will be the object of interest, that Spiri would have to follow. This simulation will allows us to test our algorithm and to experiment generating and introducing depth map.

Chapter 3

Methodology

In this chapter we present our design and implementation of an autonomous tracking-following paradigm. Section 3.1 provides a step by step implementation of the three components of the TLD algorithm, Tracking, Detection and Learning. We then show how SIFT features are introduced to reinforce the TLD tracking system. Section 3.2 describers integration of ROS framework. This in substance is setting up the network for TLD to communicate with the quadcopter. This section also provides details on setting up Gazebo and constructing a simulation environment for Spiri, leading with an illustration to show the sequence in which the data is processed and transmitted among the distributed components of our paradigm. Starting with the tracking process till the movements made by the drone. Having the TLD tracker running with SIFT features communicate with the quadcopter, section 3.3 describes the development of a control function that assists the quadcopter in performing maneuvers autonomously in reaction to the feedback from TLD on each image frame acquired from the camera. This section also gives details on how we augment collision avoidance to the control mechanism based on depth map generated by the UAV.

3.1 TLD

A simple frame-to-frame tracking mechanism by itself works under the expectation that the object-of-interest will at all time maintain a trajectory and that it will remain in the tracker's field of view. When this expectation is broken due to occlusion or disappearance of the object from the drones field of view, the tracker fails with no capability to reinitialize. In the case of tracking-by-detection discussed in chapter 2, the classifiers for object detectors have to be trained beforehand with training examples of the target object, which is not always known. Once trained tracking-by-detection approach assumes there will be no change in target appearance, which is quite restrictive and can result in failure if there happens to be any change in target's

appearance. As such, individually these two approaches in long term tracking will result in failure due to their constraints.

TLD is an object tracking paradigm developed by Zdenek et al.[18][16] for the purpose of long term tracking, that works by combining properties of the two stated approaches frame-to-frame tracking and tracking-by-detection. The tracking mechanism supports the detector by providing valuable training examples of the target object and the detection mechanism trained on these example set can reinitialize the tracker in case of occlusion or drift. The algorithm goes a step further by incorporating learning constraints that effectively estimate errors made by the detector and updates the detector to avoid such errors in the future.

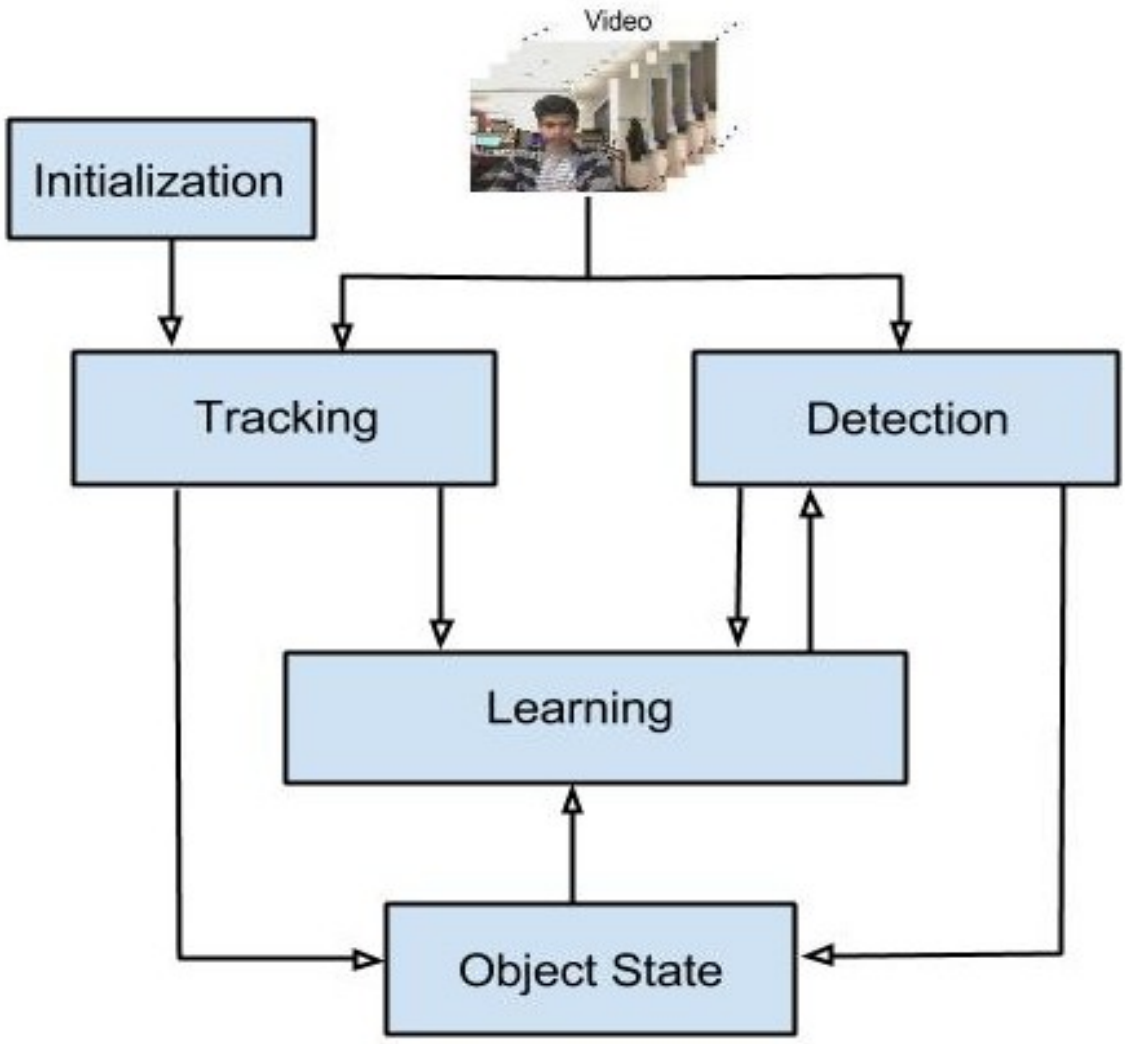


Figure 3.1: TLD block diagram

Overall, the engaging aspect of the TLD algorithm is its ability to learn new appearances of the target object without accumulating too much error to cause drift. We will apply this methodology for object tracking while modifying and augmenting it with additional components to best suit our purpose. Figure 3.1 illustrates the components of the algorithm. The remainder of this section elucidates the working of each of these components.

The first iteration of the algorithm is referred to as the initialization stage. The initial representation of the target object is set, enclosed within a bounding box. Given the initial template a new data structure is created to hold the templates of the target appearances as positive examples (P^+) and templates of the background as negative examples (P^-). This data structure is called the object model (M). The positive examples added to object model are ordered based on time. At each following iteration a single frame from the video is passed to the tracker and detector in parallel for processing.

3.1.1 Tracking

Tracking in TLD is done in 4 parts. First an equally spaced set of points is constructed spanning the bounding box. Second, the motion of each of these points is tracked by employing pyramidal Lucas-Kanade point tracking, discussed in section 2.1.1. Then, by using Lucas-Kanade tracking results two error measures are computed,

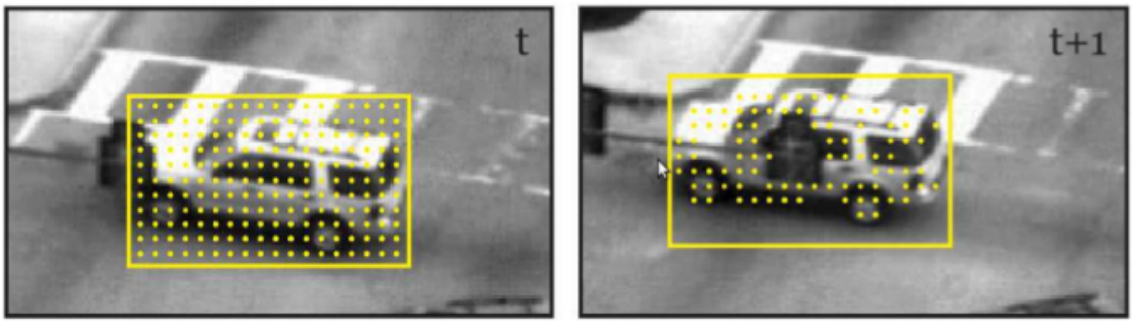


Figure 3.2: The grid points are initialized to represent the target object within the bounding box. Only the most reliable of these points will be used to calculate the transition and scale of the bounding box enclosing the target object in the next frame. [17]

forward-backward error and the normalized cross-correlation error to filter erroneous points. Finally based on a method called median-flow [18] applied on the remaining reliable points the translation and the scale change of the bounding box in the current frame is determined.

Point estimations

As shown in figure 3.2, a grid of equally spaced points spanning side to side is constructed to represent the target object within the bounding box. Lucas-Kanade point tracking is applied to individually track these points from one frame to the next. This method tracks by retaining only the most reliable points, which are corners. The points representing the regions low intensity variance within the bounding box are dropped. The reminder of the points are evaluated by the two following error measures to filter out erroneous points. This further increases the robustness of the tracker by estimating object flow based on most reliable points.

Error Measures

The first measure is called the **Forward-Backward error measure**, proposed by Kalal et al. [17]. This measure is based on the principle that, points tracked from frame t to $t+1$ should be reversible. That is, be able to track from frame $t+1$ back to frame t. This is illustrated in figure 3.3.

Lucas-Kanade applied to point 1 in image a is tracked to a point in image b. To check if tracked point is correct. Lucas-Kanade (lk) is applied again but reversed, tracking the same point back to image a. We can see that the backward tracking of the point coincides with the original location. This indicates no errors occurred and the tracked point is reliable. In the case of Point 2 however, the backward tracking points to a different location, announcing an error. The euclidean distance between the original point 2 location and the erroneous location gives us the error measure. It is defined as:

$$e = |p - p''| \tag{3.1}$$

where p represents the original location of the point being tracked and p'' represents the location of the point after being reverse tracked from image b to image a. It is as

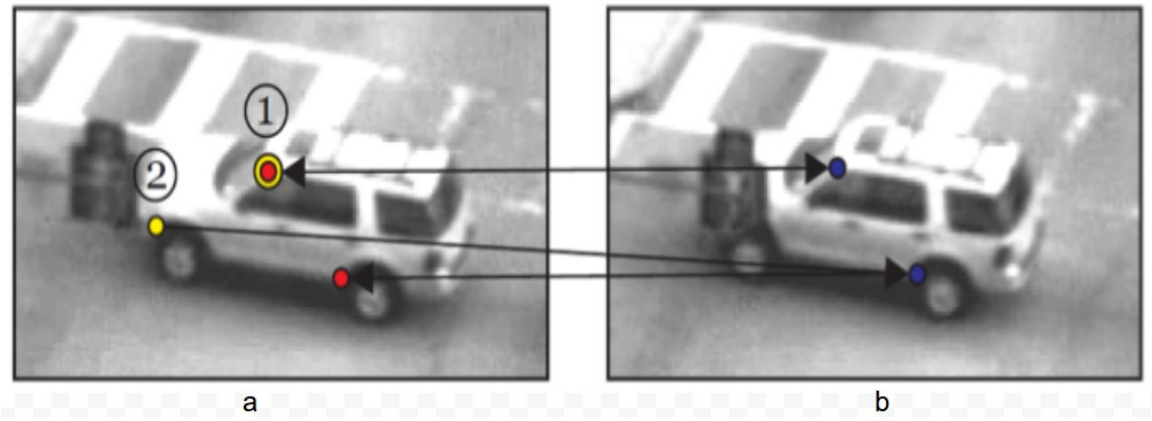


Figure 3.3: The figure illustrates the procedure for forward-backward error measure. Point 1 from image a is tracked to its correct position in b. Hence there is no error while performing its backward tracking to a. But, point 2 is tracked to an incorrect point in b. Thus, reaches an erroneous position while tracking back to image a. The euclidean distance between point 2 and the erroneous point in frame a gives us the error measure.[17]

shown in equation 3.2

$$p'' = lk(lk(p)) \tag{3.2}$$

The second error measure is based on similarity. The idea is that the patch P_1 surrounding the tracked point p should be similar to the patch P_2 surrounding the resulting point p'. The similarity of these two patches is measured using a patch matching technique called the **normalized cross correlation** (NCC) [21]. NCC is defined in Equation 3.3:

$$NCC(P_1, P_2) = \frac{1}{n-1} \sum_{x=1}^{n} \frac{(P_1(x) - \mu_1)(P_2(x) - \mu_2)}{\sigma_1 \sigma_2} \tag{3.3}$$

Where n is the number of pixels in any one of the patch. μ_1 and μ_2 represent the mean intensities and σ_1 and σ_2 represent the standard deviations of the two patches P_1 and P_2 respectively. NCC achieves robustness towards change in image brightness by subtracting the mean and scaling the standard deviation [39]. NCC returns a real value in the range of -1 and 1, with 1 indicating highest similarity between the patches

Once these two error measures are computed for each individual grid point. The median of all forward-backward error (FB_{med}) is computed along with median for all

the NCC measures (NCC_{med}). Based on the two medians the grid points are filtered to retain reliable ones. First, if the FB_{med} value is greater than a preset threshold, the whole bounding box is dropped indicating an unreliable tracking result. If the FB_{med} is lower than the preset threshold, the points with forward-backward error measure greater than FB_{med} value but with NCC measure lower than the respective NCC_{med} value is dropped, with remaining points retained as reliable.

Bounding box Transformation & Scale change

The transformation and the scale of the resulting bounding box is computed based on the remaining reliable grid points. Three parameters contribute to defining the bounding box. The vertical translation, horizontal translation and the scale change. Vertical translation of the bounding box is the computed median of all point translation in the y-direction. Similarly the median of all the point translation in x-direction gives the horizontal translation. To compute scale change, the ratio of the distance between each pair of points in the current frame against the distance between same pair of points in the previous frame is computed. The median of these ratios determines the scale change. With this the tracker draws the estimated bounding box on the current frame.

3.1.2 Detection

The object detector in TLD runs in parallel with the tracker on each frame. The detector sweeps the entire frame by applying the Sliding-Window approach discussed in chapter 2. This is to decide whether the underlying patch contains the target object or not. The objective of the detector is to essentially re-initializes the tracker whenever it fails or drifts away from the target object and to augment new appearances of the target to the object model.

The initialization of sliding-window parameters are dependent on the size of the initial bounding box. Depending on the size of the bounding box in a QVGA frame (320*240), the system can generate few $100s$ to 10^5 sub-patches to evaluate. This is in part due to the need for assessing multiples scales of each sub-patch. As explained in Chapter 2, evaluating 10^5 patches exhaustively is not efficient. Hence the classification process of the object detector is structured in three levels, cascaded one after the other.

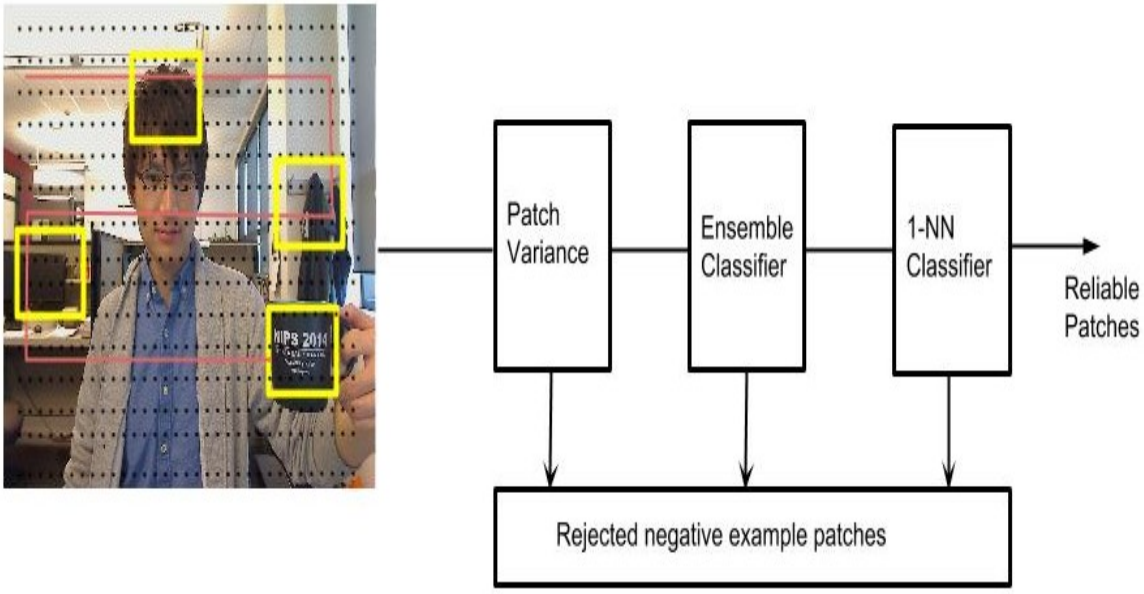


Figure 3.4: TLD detection procedure. Starting with generation of sub-patches that are evaluated by employing sliding-window technique. The light red path along the frame indicates the sliding path of the window. The reliable patches are then processed in stages. Starting with patch variance, that rejects most of the low variance background patches followed by an ensemble classifier and finally through the nearest-neighbor classier to obtain the most similar patches in the frame

Earlier stages of the cascade reject most of the non-relevant sub-patches allowing small number of patches to go through stages involving higher computations. This form of cascading improves object detection speeds. Figure 3.4 illustrates multiple stages of the detection process in TLD.

Sliding-Window

Given the initial bounding box, the detection algorithm generates multiple scales of the sliding windows varying by scale step of 1.2. These windows of multiple scales slide along the frame in the pattern shown in Figure 3.4. The horizontal shift is 10% the bounding box width and the vertical shift is 10% the bounding box height. Regardless of the initial size, each of the underlying patch is further scaled to a size of 15×15 pixels and evaluated independently.

Cascaded Classifier

As illustrated in Figure 3.4, there are three stages of the cascade. Each stage rejects unsuccessful patches from the process and passes the remaining onto the next cascade. The three stages are described as,

Patch Variance: This is the first stage of the cascade. The patches are evaluated based on their variance in this stage. The variance σ^2 is computed [43] by constructing an one dimensional vector of pixel intensities of the given patch and then applying the following equation to it.

$$\sigma^2 = \mu^2 - \frac{1}{n}\sum_{i=1}^{n} x^2 \tag{3.4}$$

where n is the number of pixels in the image patch and μ is the mean of all the pixel values in the patch, computed as shown.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3.5}$$

Patches with variance value less than a threshold are rejected. This threshold is computed as 50% of the variance of the patch enclosed by the initial bounding box. Keeping it 50% enables this stage to reject most of the uniform regions representing the background but retain structured regions for further evaluation.

Ensemble Classifier: This is the second stage in the cascade. The patches determined to be reliable by the patch variance are received as input in this stage. The ensemble classifier is made up of n base classifiers, each performing a number of pixel comparisons on the input patch. These pixel comparisons done on a patch yield binary values, as shown in Figure 3.5. These values correspond to the number of comparisons made within the classifier. A comparison of two pixels with similar intensity generates 1 or 0 otherwise. The structure of pixel comparisons are generated during initialization and remain fixed during run time. In order to maintain independence of base classifiers, each of them perform a structure of pixel comparisons different from the other. This is achieved by, discretization of pixel space within a normalized patch and generating all possible comparison sets spanning vertically and horizontally. Then, the comparisons are permuted and split among the n base classifiers.

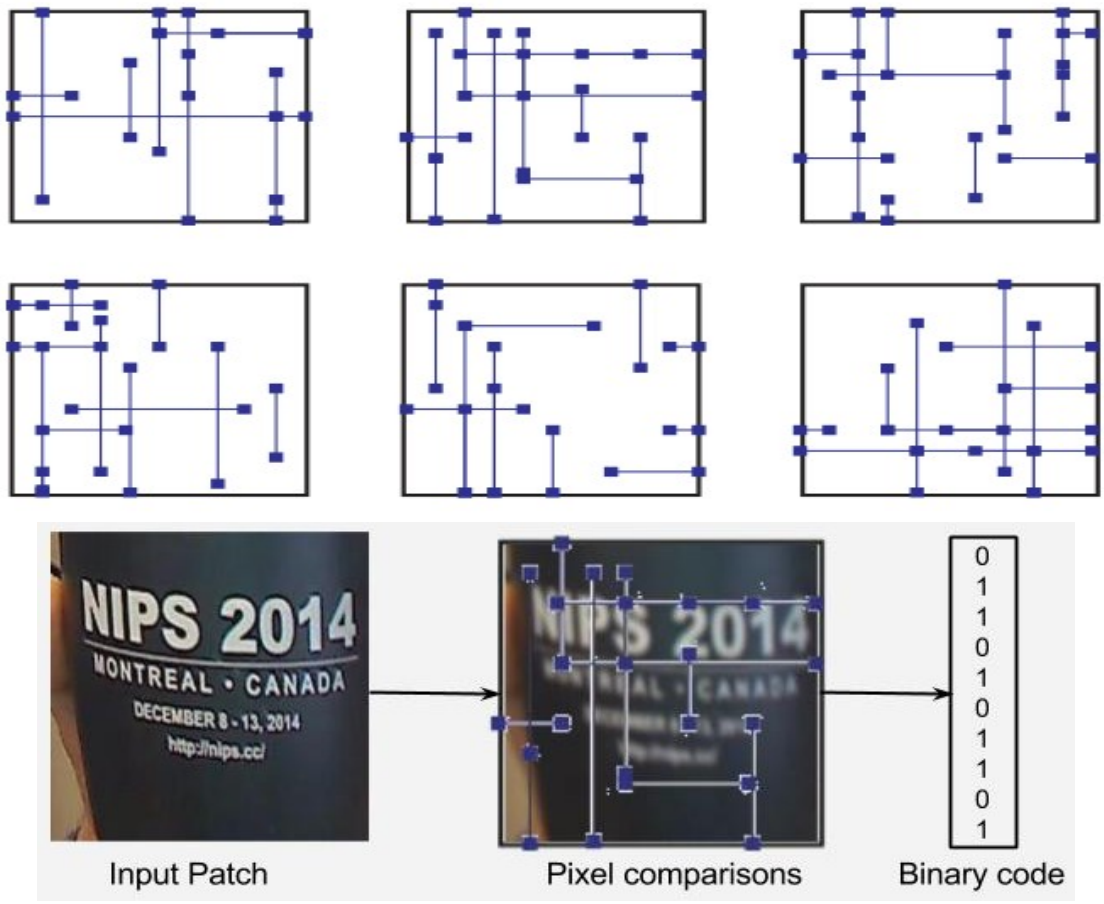


Figure 3.5: First two rows of the figure illustrate the different structures of pixel comparisons within each base classifiers. The lines connecting two rectangles within the base classifier indicate comparison. The third row depicts the pixel comparisons performed on the convolved input patch to generate the binary code

The input patch on arrival is subjected to Gaussian blur with a standard deviation of 3 pixels to improve its robustness against noise, then the comparison structure of the base classifier is stretched to fit the patch.

Each base classifier maintains a distribution of posterior probabilities $P_i(y/x)$, where $y \in (positive, negative)$ and x is the binary code generated by the base classifier. A histogram of this distribution would have 2^n entries, where n is the number of pixel comparisons. TLD in its implementation has 10 base classifiers each performing 13 comparisons. The resulting probability is estimated according to the following equation,

$$P_i(y|x) = \frac{\#p}{\#p + \#n} \tag{3.6}$$

Where, #P represent the number of positive patches with identical binary code and #n represent the number of negative patches with identical binary code. An input patch is labeled positive $P(y = positive|x)$ when the average of posterior estimation from the 10 base classifiers is greater than 50%. The posteriors of the base classifiers are updated with each new classified patch. The patches with averaged posteriors greater than 50% are passed on to the next stage in the cascade.

Nearest Neighbor Classifier: This is the final stage of the cascade. Only few patches remain after being filtered through first two stages. Patches in this stage are classified using nearest neighbor classification method. The feature space of the classifier maintains patches for both positive class P^+ and negative class P^- from the object model. The object model is itself online, updated with each iteration. As shown in Figure 3.6, the classifier, given an input patch P of unknown class label, will identify the nearest positive and negative neighbor and compute relative similarity S^r between the input patch and the two labeled patch. The classification of the input patch is based on this relative measure and is as defined in equation 3.7.

$$S^r(P, M) > \theta_{NN} \tag{3.7}$$

Here, M represents the model, that is the nearest positive and the negative patch. If the measure is greater than a manually set threshold θ_{NN} the patch is labeled positive, otherwise negative. θ_{NN} is set to 0.6, this value is defined empirically [18]. Parameter θ_{NN} can be tuned to direct the classification towards higher precision or recall depending on the application. As shown in the equation 3.7, relative similarity between the current patch and the object model defines the Nearest Neighbor classifier. The similarity S between any two patches is given by the following equation.

$$S(P_1, P_2) = \frac{1}{2}[NCC(P_1, P_2) + 1] \tag{3.8}$$

NCC as discussed in equation 3.3, yields a result ranging from -1 to 1. Equation 3.8, simply computes NCC and reports a value between 0 and 1, with values towards

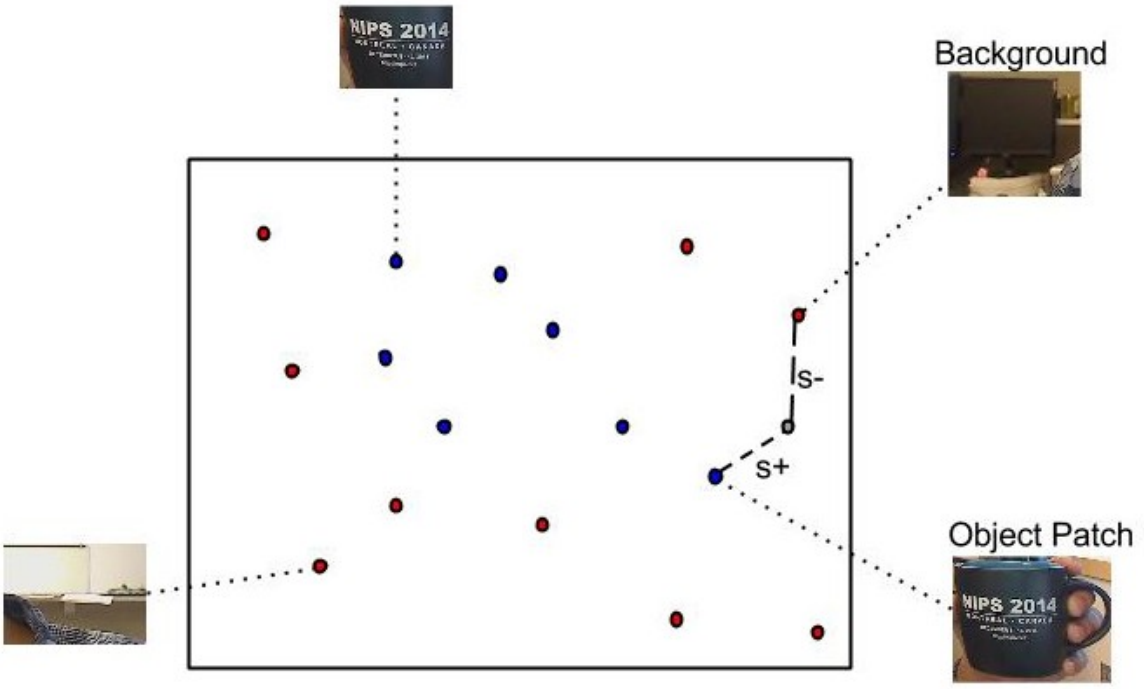


Figure 3.6: Figure illustrates the appearance space used to perform the nearest neighbor classification. The red dots represent negative patches and the blue dots represent the positive object patches. The gray dot depicts a patch in need to be classified. S^+ and S^- indicate the similarity measure with the closest positive and negative patches respectively

1 indicating high similarity. In order to compute relative similarity established in equation 3.7 for a given input patch the following formula is applied.

$$S^r(P, M) = \frac{S(P, P^+)}{S(P, P^+) + S(P, P^-)} \tag{3.9}$$

Relative similarity yields values between 0 and 1, with 1 representing highest confidence that the contents of the patch depicts the target object. The patches that are classified as positives are then sent to the component Object state for integration with the results obtained from the tracker.

3.1.3 Learning

The learning component of TLD performs two tasks, The first task it to train the detector during initialization with initial training examples. The second task is to augment the detection component in each iteration of the algorithm to continually

improve its classifiers performance. As discussed in section 2.3.2, this is done by analyzing results of the detector and enforcing two constraints on them called the P-constraint and the N-constraint.

During **initialization**, the classifiers are trained by synthesizing the single initialized bounding box of the target object to generate more positive training examples. To realize this, a batch of 10 patches overlapping the initial bounding box is selected from the scanning grid. For each of the selected patches, 20 warped version are generated by performing shift of $\pm 1\%$, scale change of $\pm 1\%$, in-plane rotation of $\pm 10^{\circ}$ along with gaussian noise, $\sigma = 5$. The remainder of the background patches in the initial frame are collected as negative examples. This labeled set of examples form the initial object model and the initial training set to train the detector.

The task of **P-constraint** is to generate positive examples by identifying false negatives classified by the detector based on the trajectory of the target object. The trajectory is established based on confident results from tracker and the detector. Following that a subspace is created around the trajectory defined by a threshold θ_L. θ_L defines growth of the subspace, setting it to a low value creates a subspace that is more stringent to growth and hence reducing learning of new patches. But, a high value would allow the subspace be more liberal with its growth and hence accumulates erroneous patches, affecting classification. In the implementation, $\theta_L = 0.7$ is set empirically.

When the tracker moves within the constructed subspace the patches are considered reliable and any occurrence of negative classification by the detector would be considered as false negative. The subspace is extended to grow, only if the tracker moves out from within the subspace. In the event of tracker reinitialization, the sub space is also reinitialized. If there is an immediate generation of bounding box by the tracker outside the subspace without any continuous trajectory. The tracker is reported to have made an error and is re-initialized by the detector.

The task of **N-constraint** is to determine false positives classified by the detector and thereby improve the classifier with each iteration. As discussed in Section 2.3.2, the identification of false positives is based on the assumption that the target object can occupy only one location in an image frame. Occurrence of multiple bounding box as positive examples of target object in a frame is considered false and hence only

the bounding box highly overlapping with the location of the bounding box from the previous frame is considered as positive with the rest relabeled as negative examples. Both P-constraint and N-constraint are enforced simultaneously on the negative and positive results respectively, generated by the detector in the same iteration of the algorithm.

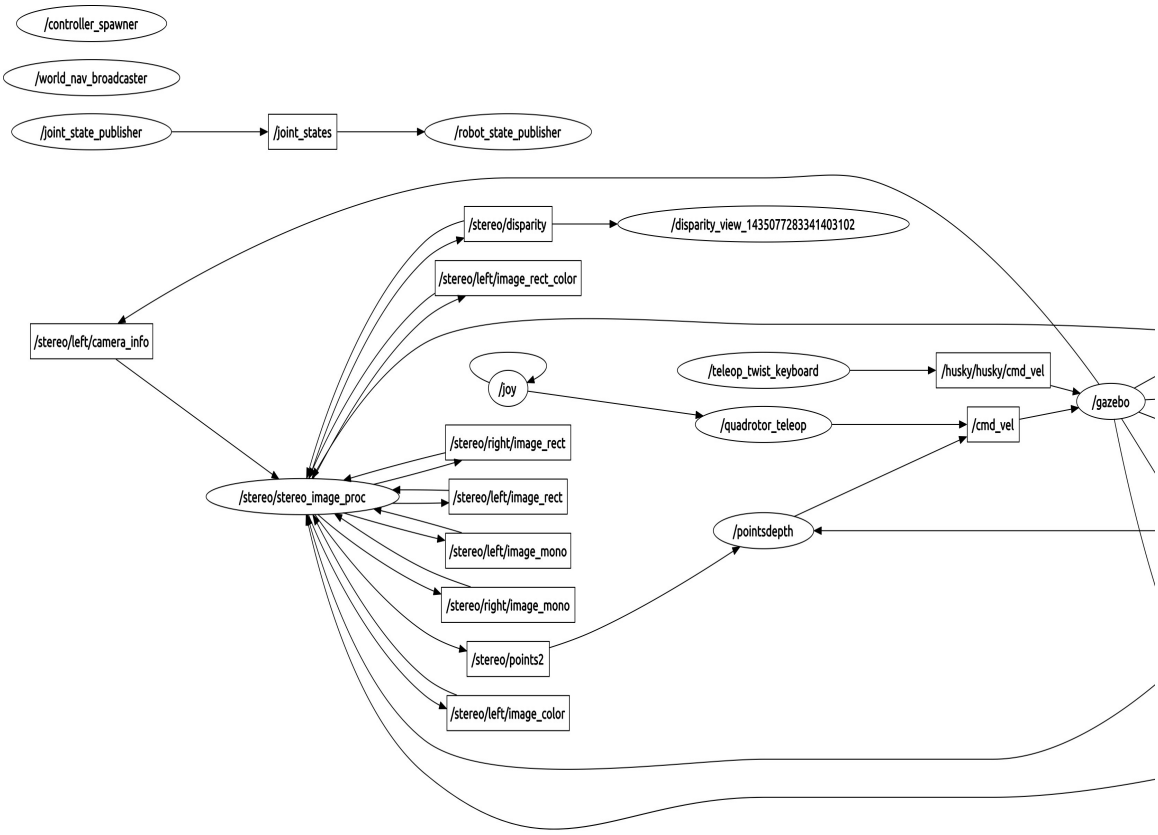
3.1.4 Object State

The component Object state as illustrated in Figure 3.1 receives results from both the tracker and the detector. The component's task is to analyze the results and report the most confident bounding box for the current frame. The detector presents multiple bounding boxes with varying confidence and the tracker provides a single bounding box. When both detector and tracker have no output the target is reported as lost. The usual case, has multiple results from detector overlapping the bounding box from the tracker. In such instances, the detector generated bounding boxes that have greater than 80% overlap with the tracker bounding box is averaged to generate a single bounding box for the current frame. The tracker is re-initialized, if its generated bounding box is far from the maximally confident detector generated bounding box .

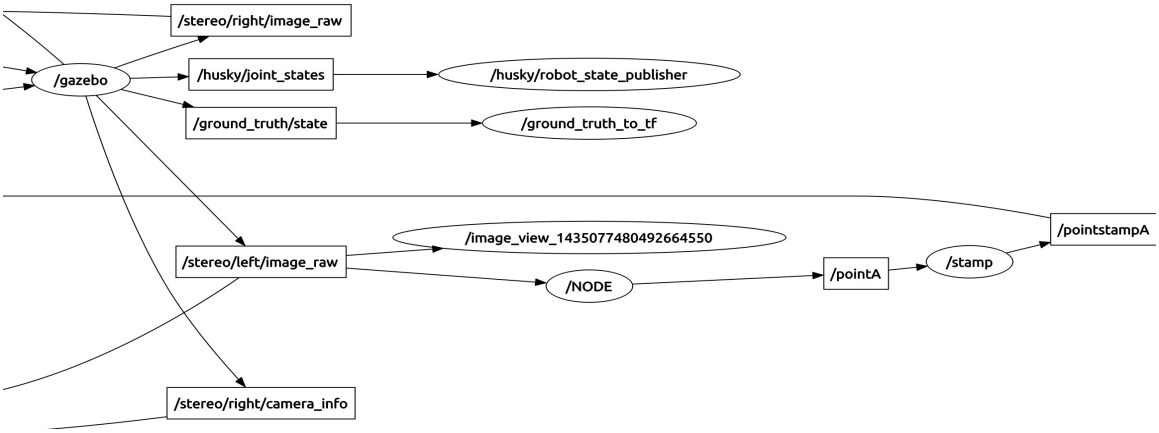
Thus, after generating the bounding box for the current frame. Next iteration of the algorithm is commenced on the arrival of consecutive video frame. To conclude, TLD algorithm is recursive with aforementioned processes running indefinitely, tracking the target object until interrupted by the user.

3.2 ROS-TLD integration & Simulation setup

ROS as discussed in Section 2.6, is the framework implemented to facilitate communication between various components of our tracking and following system. The tracking algorithm is modified to support ROS, which allows it to subscribe to drones camera feed and publish the tracking results to other components of the system. Also, the data acquired from the camera feed is processed by an open-source computer vision library called OpenCV [3] to convert the raw data into a format presentable to the tracking algorithm. OpenCV programming functions are also extensively applied in TLD to perform necessary image manipulations.



(a) RQT graph



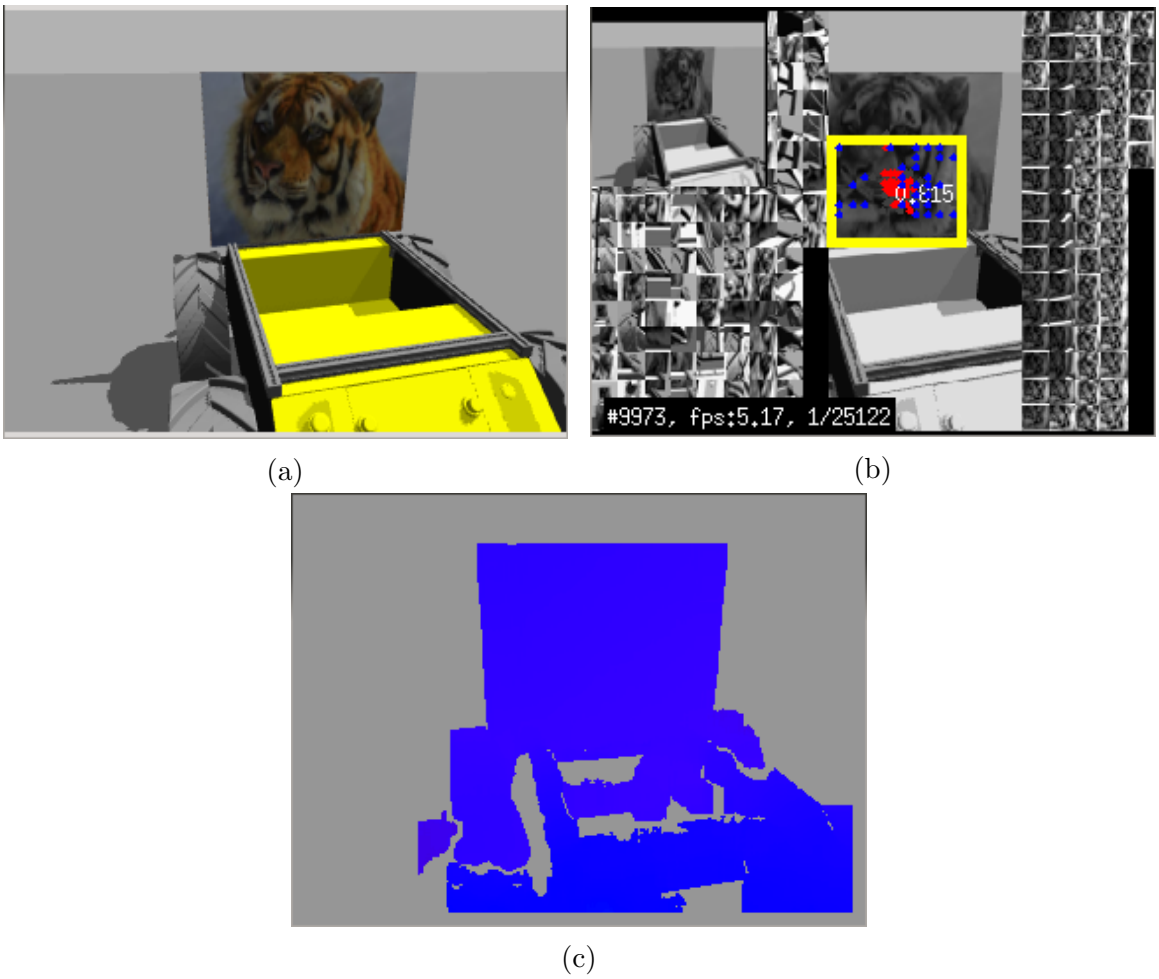
(b) RQT graph, Continued.

Figure 3.7: RQT graph, illustrates all the components and their active communication during runtime. The nodes are represented by ovals and topics are represented by rectangles. Line arrow indicates communication between two nodes through a topic. The node at the tail is publishing data while the node at the head of the line arrow is subscribing to that data.

Figure 3.7 displays the RQT graph of the whole system, containing the topics and nodes that are set-up and executed to successfully complete object tracking and following of a target by an UAV. We shall describe the significant processes shown in the graph that take place during runtime. It is important to revisit that, nodes are represented by ovals and the topics are represented by rectangles. Communication between two nodes are indicated by a line arrow connection between them through a topic.

We start with the node */gazebo*, this is the simulation program that enables the two model robots. When real robots are utilized for the setup, this node would simply be replaced by nodes enabling the actual robots. Model Husky is the target to track and follow and Model Spiri is our UAV. We can see multiple topics being published by and subscribed from gazebo. Topic */husky/joint_states* for instance defines the appearance of model husky, it's joints and visuals. The state nodes for Spiri isn't visible in the graph. The reason being, its properties have been integrated within the node gazebo. But, we do see its topics being published from gazebo like */stereo/right/image_raw*, */stereo/left/image_raw* and also subscribing to topics like */cmd_vel* among others. /gazebo also subscribes to topic */husky/cmd_vel* published by node */teleop_twist_keyboard* . This topic caries commands from the user to remotely maneuver husky in the simulation world.

The node named */NODE* represents the TLD algorithm. It subscribes to camera topic */stereo/left/image_raw* published by Spiri from the node */gazebo*. The data is then processed in TLD to generate the resulting bounding box. This bounding box location data is then published through the topic */pointA*. In Figure 3.7 we can see a topic */pointstampA*, that carries bounding box data that is stamped with the time internal to ROS, from the node */stamp* all the way back to the node */pointsdepthA*. This node is the following mechanism that maneuvers the drone by generating necessary commands. It also acquires depth information by subscribing to the topic */stereo/points*2. When its done processing all the information, it generates the necessary velocity commands for the drone and publishes it over the topic */cmd_vel*. Node */gazebo* subscribes to them and conveys it to the drone. This chain of nodes and topics together successfully accomplish tracking and following.



(a) (b)

(c)

Figure 3.8: Different views of the target object husky in simulation, (a) This is the left camera view from the drone facing the target (b) The view from TLD, it shows the stored patches along with bounding box enclosing target features (c) This figure shows the depth map generated using Spiri's stereo camera. This blue color object is made of individual depth points, indicating proximity of the object.

Figure 3.8, illustrates view of different image data at various stages of the algorithm. Figure 3.8a is the raw image data that has been properly formatted by OpenCV library functions. Figure 3.8b is the view inside TLD, generating the bounding box result. Figure 3.8c depicts the depth map of Husky generated from the topics published by Spiri's left and right camera and processed by the node */stereo/stereo_image_proc*.

We use a popular open-source library called the Point Cloud Library (PCL) to generate the depth map information, required by our tracking system. PCL is primarily applied to perform 3D geometric processing in computer vision [32]. The node

/stereo/stereo_image_proc given both the camera feeds from the stereo camera, contains the necessary functions to calculate the disparity between the frames of the left and right camera. Based on this disparity it generates a three dimensional points. Each point mapped to the 3D world space contains the respective depth information. These points together in an image frame is referred to as the Point Cloud. The Point Cloud for each frame is published over the topic */stereo/points*2. In the next section, we shall describe the methods applied to achieve target following.

3.3 Target following mechanism and collision avoidance

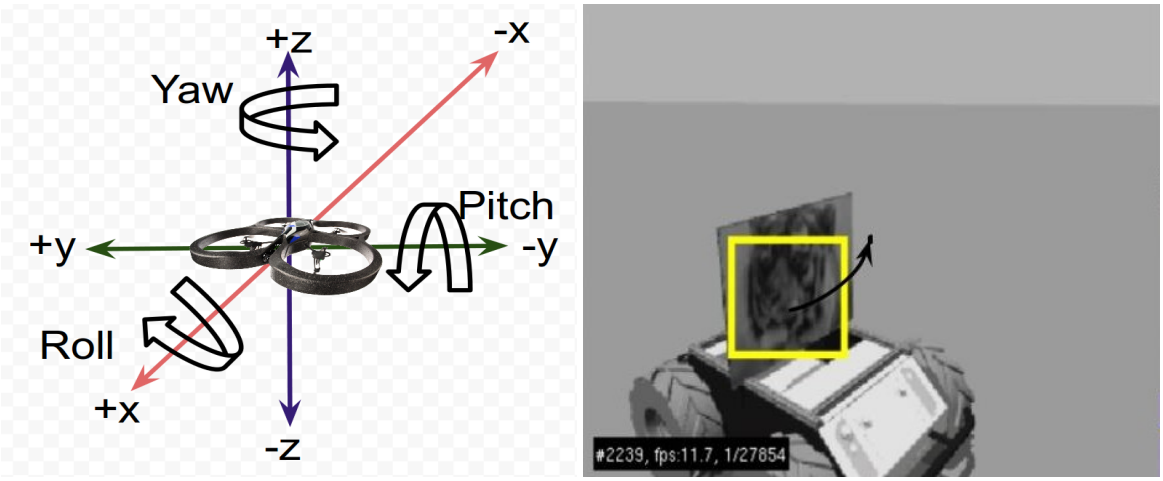
An UAV moving freely in three dimensional space has six degrees of freedom. Three translational and three rotational movement about the three perpendicular axis as shown in Figure 3.9a. The rotations about the axis are referred to as yaw, pitch and roll. UAV can be maneuvered by varying them while providing linear acceleration. Along with its movements, UAVs have to perform other basic maneuvers like taking-off, landing and hovering. In AR.Drone and Spiri these maneuvers are automated and can be initiated with a simple command. The AR.Drone controller supports controlling the drone by varying velocity along a particular axis, that gets automatically mapped to necessary yaw, pitch and roll angles.

The objective of the target following mechanism is to mimic the movement trajectory of the target. Hence, given the target location in UAVs field-of-view by the tracking algorithm. The mechanism needs to retain the object at the center or maneuver the UAV appropriately to position the target to the center of UAVs field-of-view. As the bounding box generated by TLD encloses the target. The center pixel coordinate of the bounding box should coincide with the center pixel coordinate of the image frame. The image frame mentioned here is the same as the UAVs field-of-view. If the target pixel location is already at the center of the frame. No action is required by the UAV and it can simply be made to hover at the same location. When the target starts to move, the mechanism on each frame of the camera feed computes the location of the bounding box center and determines the displacement from the frame's center in both the X and Y axis, as shown in Figure 3.9b. The two measures, displacement and location refers to pixel coordinates of the image frame. Figure 3.10b illustrates end to end coordinate locations in an image frame of size 320×240.

For instance when the mechanism encounters target displacement of 150 pixels on X axis from the center. It should immediately initiate the UAV to move at certain velocity towards the positive X direction to off-set the displacement made by the target. A mapping function is used in order to generate the appropriate velocity given a pixel displacement value. Higher displacement results in higher velocity value. The mapping function applied in our implementation is from the Arduino library [24], it is as illustrated in equation 3.10

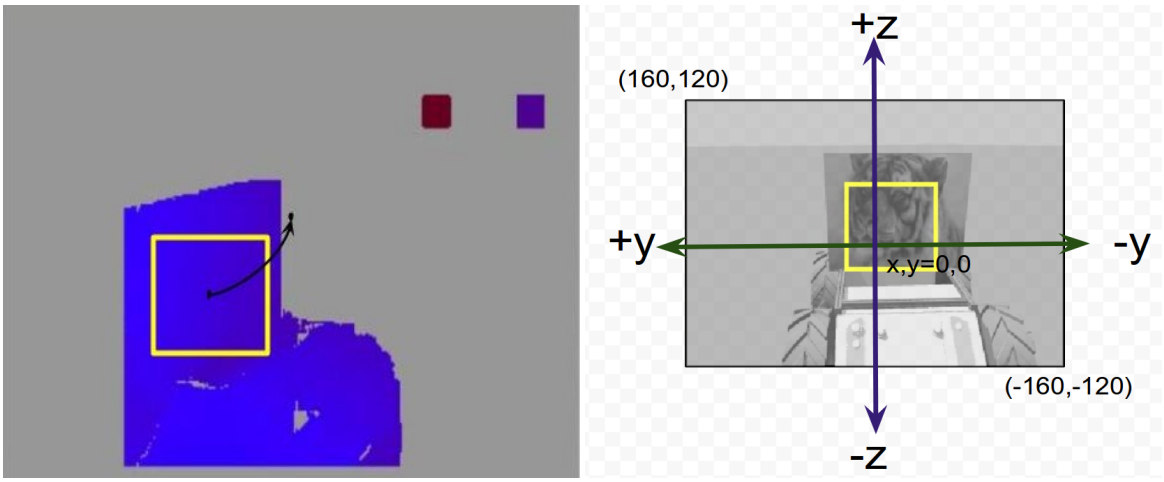
$$f(x) = \frac{(x - in\_min) * (out\_max - out\_min)}{(in\_max - in\_min)} + out\_min \tag{3.10}$$

Where x is the pixel location of the target for which we want a velocity value, The terms in_min and in_max refer to the range of our input pixel value. According to the width of our frame, $in\_min = 0$ and $in\_max = 160$. Terms out_min and out_max refer to the range of output velocity values. We have $out\_min = 0$ and $out\_max = 1$. Hence, UAV can go up to a maximum speed of 1 m/s. This formula can handle reverse range of values as well, that is negative axis values initiate reverse acceleration of the



(a) (b)

Figure 3.9: (a) Illustrates the 6 DOF, Linear velocity along X axis propels the drone forward, along Y axis generates lateral movement and along Z to increase or decrease altitude. Applying velocity along $+X$ axis decreases the pitch allowing the drone to move in forward direction (b) The target has moved away from the center of the frame. The drone would make appropriate maneuvers until the target bounding box is put back at the center.



(a) (b)

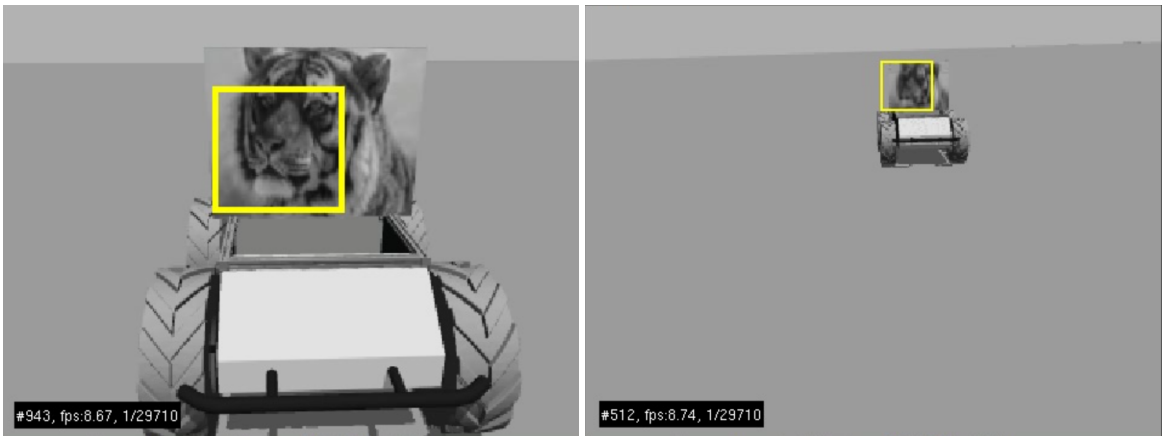
Figure 3.10: (a) Displays the depth map of UAV's view. The target in the depth map is made up of individual points, each containing a particular depth information. These points together is referred to as a point cloud. (b) Illustrates the pixel coordinates of the image frame. The frame center is $(0, 0)$. Also shows the maximum and minimum pixel range from top-leftmost point to bottom-rightmost point.

UAV.

This velocity value is then put in proper ROS messaging structure and published over the topic */cmd_vel*. The UAV listening to this topic, executes the velocity command on arrival of the message data. The target following mechanism constantly generates velocity commands to maneuver UAV for each consecutive video frame until the bounding box is pushed back to the center of the image frame.

The mapping so far covers UAV's Z axis, its altitude and the Y axis, its yaw. But, in order for the drone to maintain a fixed distance from the target. The drone needs to know how far the target is from itself to initiate appropriate velocity along its X axis (refer Figure 3.9a). That is, its forward pitch. Maintaining a stable distance to the target object is the first step towards collision avoidance. Two methods have been implemented, to estimate and maintain a constant distance to the target. The first method is applicable to UAVs with monocular camera. In such cases, distance to the target object also referred to as its depth, has to be approximated with just the bounding box properties.

From Figure 3.11, it can be seen that a target closer to the UAV has a bigger bounding box relative to a target farther away. The reason for this is perspective.



(a) (b)

Figure 3.11: Figures illustrate the relationship between size of the bounding box and distance to the target (a) The bounding box at 1 meter distance, covers a large portion of the image frame (b) bounding box at distance greater than 10 meters, covers a small portion of the image frame

The dimensions of an object closer to the camera appears bigger and occupies larger portion of the cameras field-of-view. From Section 3.1 we know that, the euclidean distance between the keypoints on the target determines the dimensions of the bounding box. Hence, a closer object due to cameras perspective looks stretched causing the euclidean distance between the keypoints to increase. This results is a larger bounding box. Adopting this property of bounding box, the mapping function from Equation 3.10 is applied to generate velocity commands along X axis based on the size of the bounding box. This maneuvers the UAV forward towards its target to maintain constant distance.

An alternative method became necessary, as estimating depth from bounding box generated by an adaptive tracking system proved to be problematic. The complications are discussed in the Chapter 4. A reliable and computationally non-demanding alternative method for estimating depth was to employ a stereo camera. An UAV integrated with stereo vision will be capable of generating a depth map of its scene online. Figure 3.10a shows the depth map generated by the simulated UAV integrated with stereo camera.

Going back to the RQT graph in Figure 3.7, the node */stereo/stereo_image_proc* is responsible for generating the depth map and making it available to the drone at each frame. The depth information from points that lie within the bounding box

determines the distance to the target. The target following mechanism applies the mapping function to generate appropriate velocity to constantly maintain the target at fixed distance. In our implementation this fixed distance is set to 1 meter. The mechanism also keeps track of overall minimum depth in the current video frame. If an object gets closer than the set overall minimum distance. The mechanism immediately prioritizes collision avoidance and maneuvers the UAV backwards until the set minimum distance is clear. The following chapter provides discussions on observation and results from our real-time and simulated implementation of our algorithm.

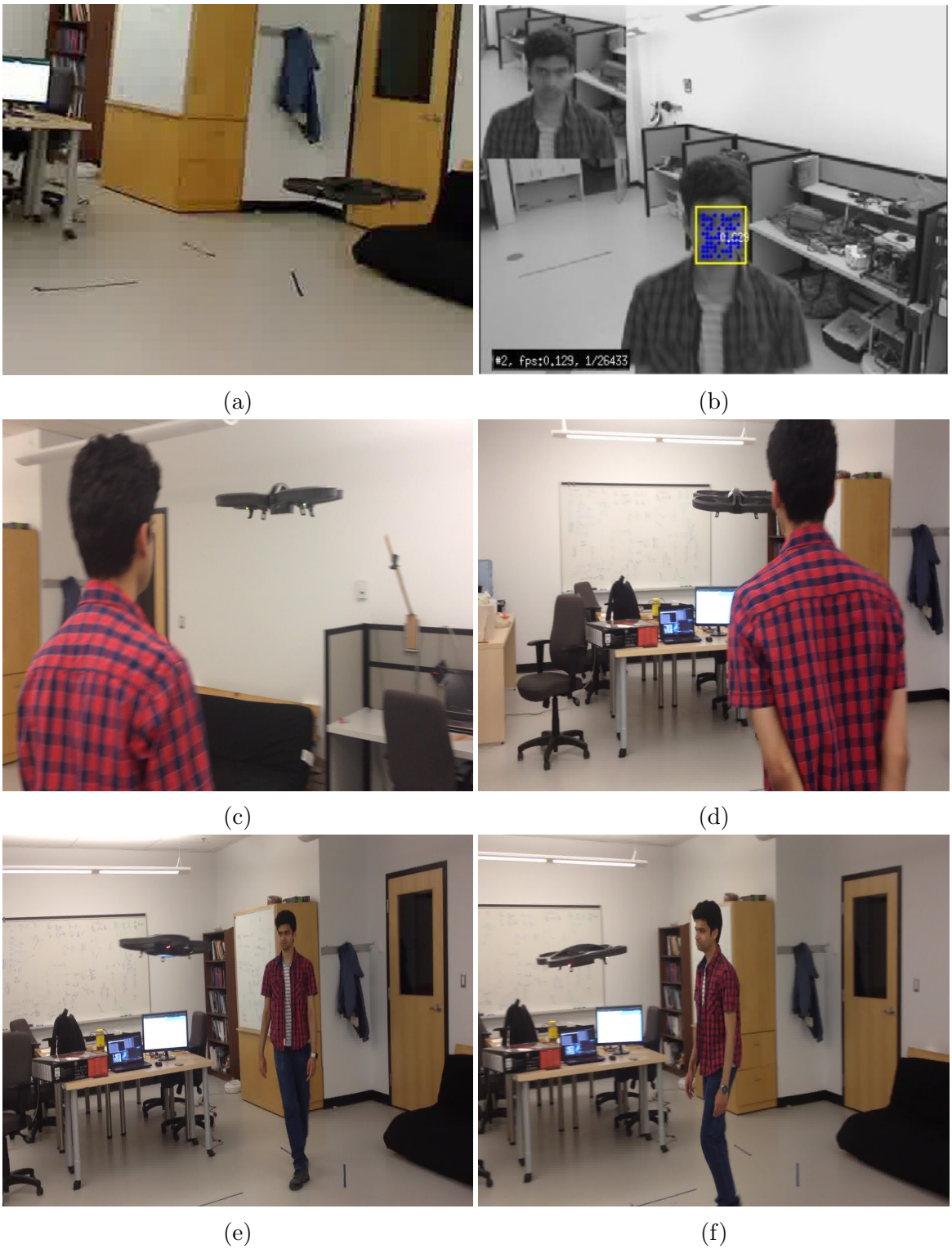
Chapter 4

Observations and Results

In this chapter we demonstrate our implementation and discuss the experiment results. In Section 4.1 we give illustrations of the tracking-following paradigm in action with a brief summary of what was successful and the observations made from the behavior of the drone. In Section 4.2 we present the results from the experiments conducted in simulation with characteristics similar to the real-world demonstration with the AR.Drone and provide discussions on the behaviors noted in Section 4.1 with illustrations. Similarly in Section 4.3 we illustrate results obtained through experiments done in simulation with depth information. In Section 4.4 we discuss about the dataset used to empirically validate our algorithm along with details on the selected evaluation metrics. The dataset used is both from the literature and those recorded during experiment. Lastly, Section 4.4.2 appraises the results obtained with the introduction of SIFT features and depth information to the algorithm, comparing it with traditional TLD with no depth information.

4.1 Demonstration

The tracking-following paradigm was applied to the quadcopter and deployed successfully for real-world demonstration. Figure 4.1 describe the progressive stages during demonstration. Starting from lifting off to landing. The Figure 4.1a shows the drone hovering, waiting for the tracker to be initialized. The user then initializes tracking by drawing a bounding box over the target visible in an interface on the host system. This requires the target to be within the drone's field-of-view. Once initialized the drone starts to autonomously follow the target. The target object is shown in Figure 4.1b. The process runs continuously until interrupted by the user. In our implementation, in the event of object occlusion or when the target has escaped the field of view the drone goes into hover mode. This action can be replaced to perform a task to search and recover targets position depending on the application and its



(a) (b)

(c) (d)

(e) (f)

Figure 4.1: Tracking-following in action: (a) Drone in hover mode, waiting to be initialized (b) Target initialized in an interface on the host system (c) Drone maintains the target at center of its field of view and at a fixed distance of 1m (d) Drone tracking and following the target autonomously (e) Depicts its forward acceleration to maintain target distance (f) Depicts drones reverse acceleration.

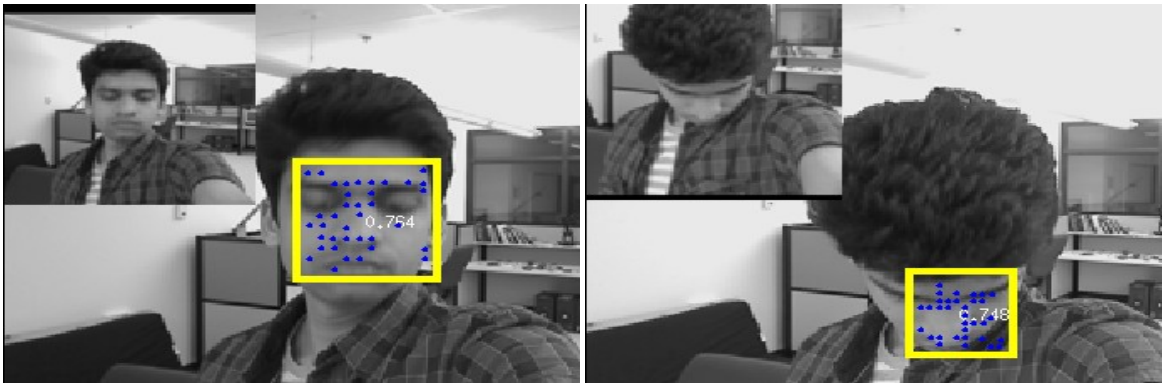
environment.

4.1.1 Observation

The drone does follow the target, but falls short at maintaining a stable distance to the target. This is clearly noticeable as the drone tends to have immediate erratic movements and considerable oscillations as it tries to maintain a fixed distance to the target. There are two prominent factors jointly responsible for causing oscillation.

- As discussed in the implementation, the drone with its monocular camera depends on the bounding box enclosing the target to formulate distance. The distance is mapped proportional to the area of the bounding box. Hence, even the slightest variation in the bounding box size is interpreted as change in distance causing the drone to oscillate.

- TLD is an adaptive tracking system, learning features of different appearances of the target object. Certain appearances of the target can have richer or fewer features to track. The size of the bounding box is based on the number of robust features of the target appearance being tracked. As shown in the figure its possible for the target object present at the same position to change its appearance and consequently affect the size of the bounding box causing the drone to overshoot and oscillate.



(a) Bounding box area = 3476 pixels (b) Bounding box area =2321 pixels

Figure 4.2: Bounding box size varies based on the features being tracked on the current target appearance. The target can vary its appearance while resting at the same location. The dots within the box mark the features.

Due to these factors, the bounding box approach to mapping distance was dropped.

Subsequently, a reliable, inexpensive and computationally non-demanding method to determine depth was to upgrade the monocular-camera acquisition to that of a stereo-camera. The merits with regard to object tracking and following are illustrated by comparing the results in following sections.

4.2 Simulation Results with bounding box

This section presents empirical results obtained from applying our algorithm to the quadcopter Spiri in a simulated environment. The experiments for this section were constructed to gauge the effectiveness of the target following mechanism at maintaining a set distance from the target and to avoid overshooting or collision. The drone tracks and follows its target, Husky. The process of experiment is to start the target from a resting state with the drone already initialized and hovering at a set distance

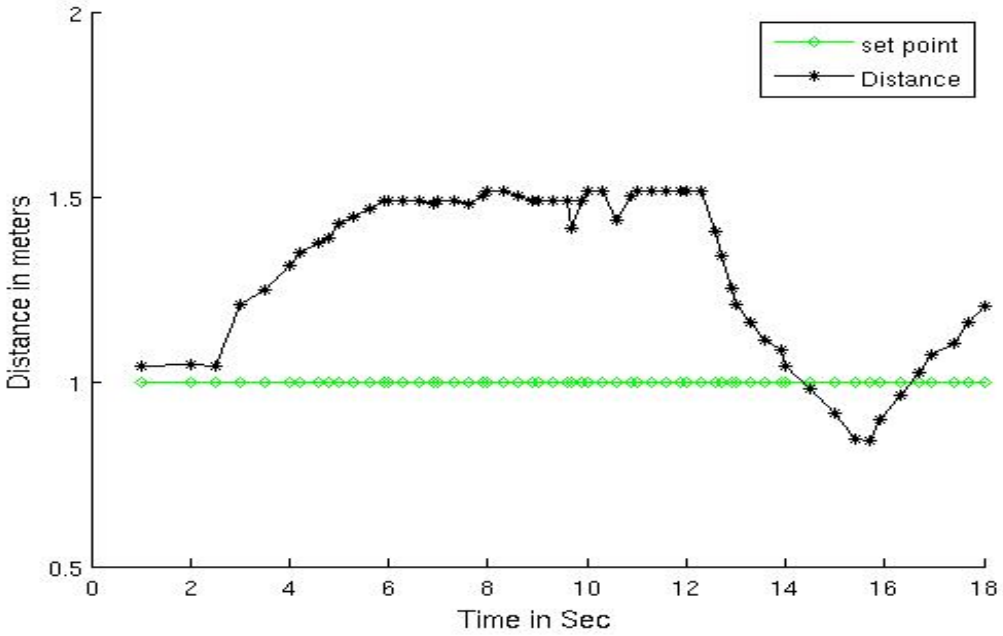


Figure 4.3: This plot depicts erratic variations in the distance between the drone and the target against experiment time measured in seconds. The sudden movement of the target at time 3 increases the said distance until stabilizing to a moderate extent about 50 cms off the set value. The sharp variations at time 10 and 11 is due to changes in the bounding box size, a consequence of change in target appearance.

from the target. The target then moves forward in a straight path and stops immediately after 12 sec. A good following system would mimic the targets movements to retain the target at the center of its view while maintaining the set distance. The set distance to maintain is 1 meter. The plot in Figure 4.3 illustrates the problems stated in the observation.

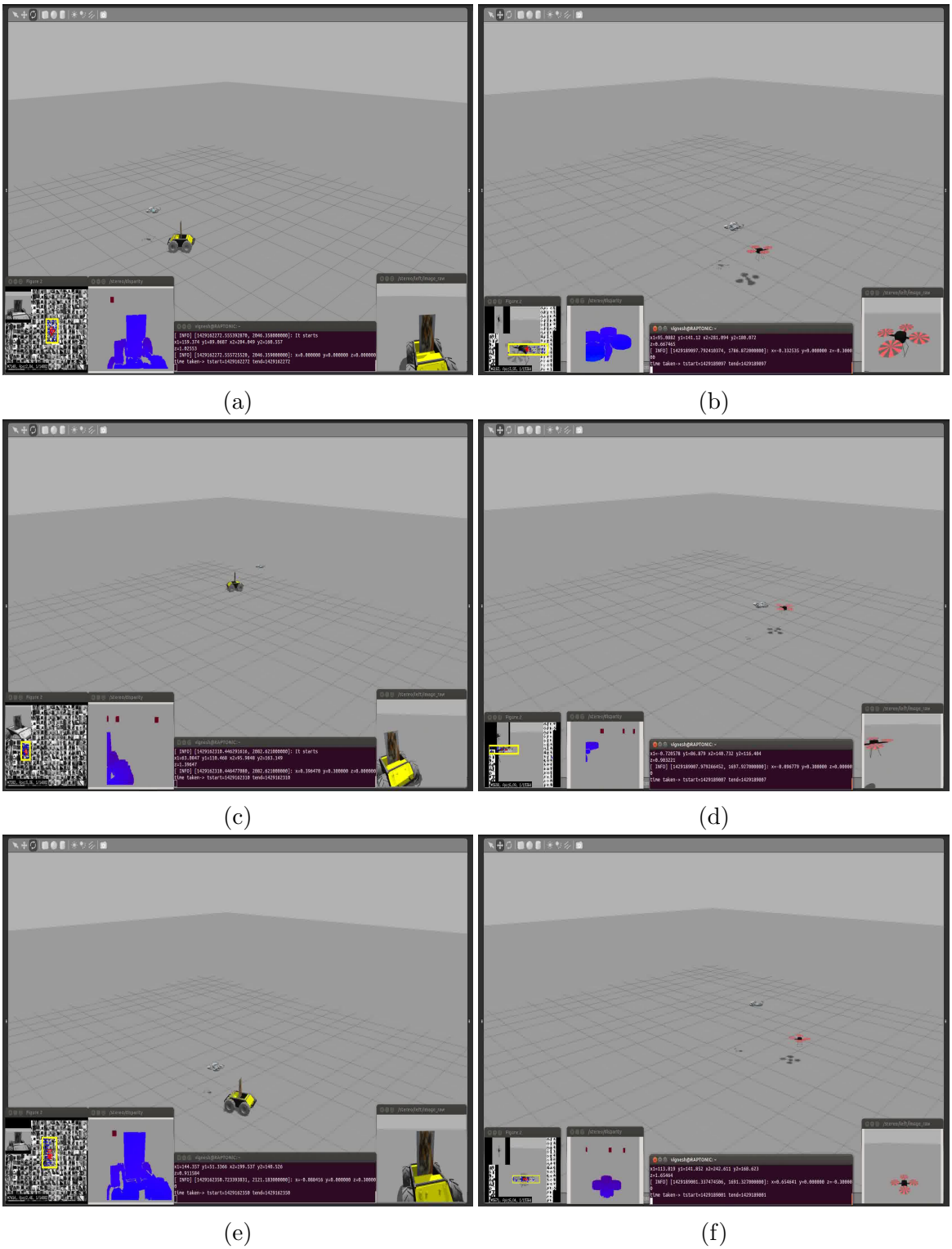
We can see that the drone did not respond swiftly to the immediate target movement at the beginning leading to accumulation of considerable gap between the target and the drone. This is in part due to how the bounding box dimensions are generated. We know that, the dimensions of the bounding box is dependent on the keypoint positions on the current target appearance. A simple change in appearance can cause huge variations in size. With that said, Its also possible for the bounding box to remain same or have a very small variation to changing appearance, causing the drone to believe the object is still in the same position for small period of time.

Another interesting characteristic of the result is the conflicting variations at time 10 and 11 secs from the start. The drone for a fraction of time believes that the target has increased its velocity and is moving further away and hence it increases its velocity to offset the raising distance, but then instantly receives another input conflicting with the previous input and hence decreases its velocity. This occurs twice in the interval of 1 sec. This is again due to the change in appearance of the target while still moving at a steady pace.

The drone moderately stabilizes between the interval of time 5 and 13 as the bounding box size maps to 1m distance. Resulting in a error of 50cms. The last point to notice is the overshoot at time 15. In the event of a sudden halt by the target a offset like this could cause collision. In real-world application this could prove to be detrimental.

4.3 Simulation results with depth information

This section presents results obtained by performing experiments on our algorithm in a simulated environment employing a model of the quadcopter Spiri. The new aspect here against the model used previously is the inclusion of depth information from Spiri's on-board stereo camera. Similar to Section 4.2, the experiments for this section are constructed to gauge the effectiveness of the following mechanism at



(a) (b)

(c) (d)

(e) (f)

Figure 4.4: Tracking-following paradigm in simulation: (a) Spiri an UAV tracking and following Husky an UGV (b) Spiri tracking and following another drone (c) Making a sharp turn while following the target husky on ground (d) Making a similar turn while following the target drone in air (e) Maintains fixed distance even at immediate halt of the target (f) Spiri slowly increases altitude when the target starts to accelerate, to retain its view of the target

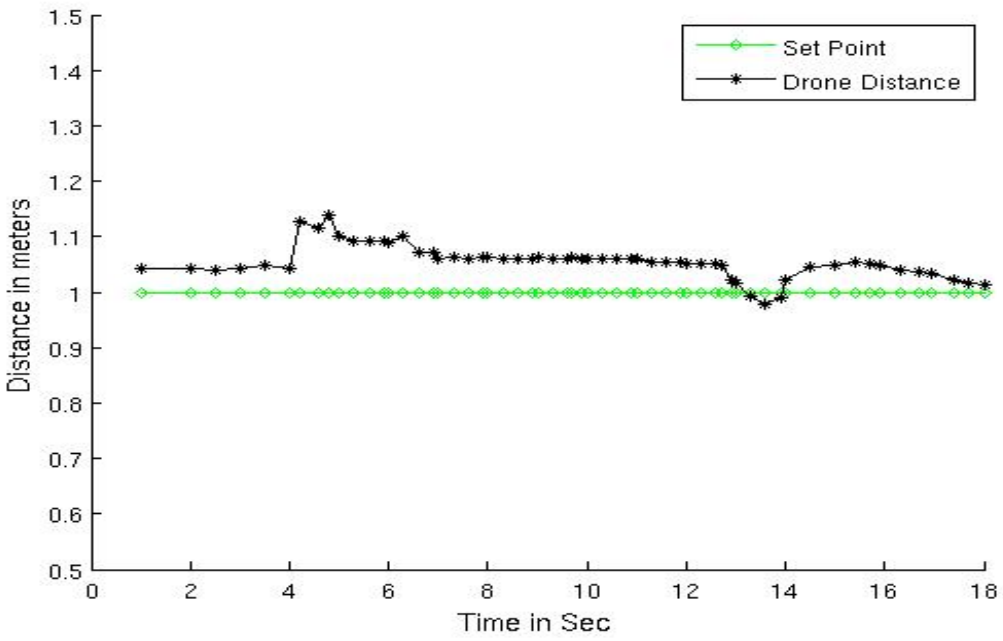


Figure 4.5: This plot depicts the distance maintained between the drone and the target Husky. We can see that the drone is more responsive with smoother transformation to target movements with the inclusion of depth information from its on-board stereo camera. The gap introduced at the sudden target movement is quickly offset and the overshoot has been drastically reduced to less than 5 cms.

maintaining a set distance to the target in order to avoid overshooting or collision and to compare the results to state the merits. Figure 4.4 illustrates the simulation environment.

The target is the UGV Husky, shown in Figure 4.4e. It is maneuvered using a joystick to engage the drone to follow. The process of experiment is to start Husky from a resting position along a straight path and stop immediately after 12 sec. Maximum velocity attainable by both the vehicles is set to 1 m/s. Distance between them during the process is collected. Figure 4.5 plots the results. The plot is evident of much better results in comparison with results from previous section. The response of the drone to the sudden movement of Husky is good. The immediate rise in distance is quickly offset and drone remains in smooth transition with the movements of Husky. Conflicting variations noticed previously are no longer present. Most importantly the overshoot noticed previously during the immediate halt of the target has been drastically reduced. Overall, the incorporation of depth information from

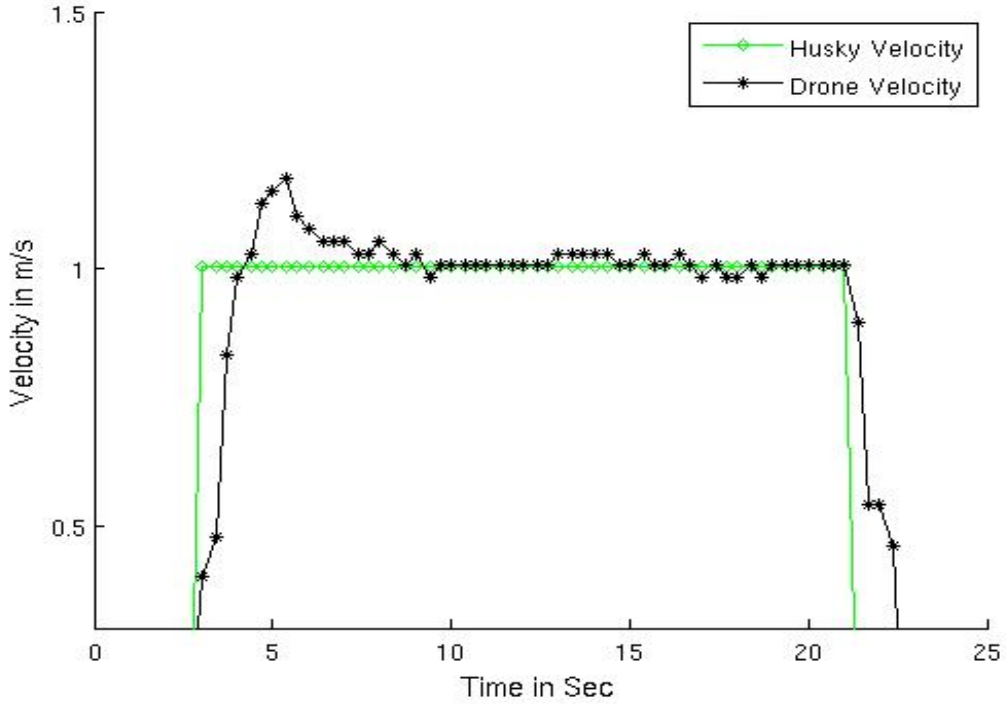


Figure 4.6: Plot illustrates velocity comparison of Husky and Spiri. Results obtained from simulated experiment. Husky is maneuvered along a straight path with varying appearances moving at a constant speed of 1 meters/sec. The plots illustrates swift response by the following mechanism enabling Spiri to follow Husky's movement.

stereo camera generates valuable distance measure needed to perform target following with smooth transition. Figure 4.6 illustrates the velocity comparison between the drone and husky. Since husky is maneuvered by the user using keyboard, it has constant linear velocity of 1 meters/sec. We can see the drone accelerate promptly at the sudden movement of Husky. It off-sets the additional distance and smoothly aligns itself to the same speed as that of Husky for smooth target following. The same effect can be observed when husky comes to an immediate standstill. Hence, the incorporation of depth information through stereo camera remarkably enhances the target following mechanism providing swift and smooth transition without much increase in computational cost. The following sections, evaluate the performance of our algorithm.

4.4 Dataset and Evaluation metrics

In order to evaluate our tracking algorithm and compare results with TLD, we have used several frame to frame video sequences from the literature [15] [46] along with a new sequence that accentuate the problems stated in previous section. In these video sequences, a specific target has been manually annotated with bounding box. A tracking algorithm initialized on the same target is ideally expected generate bounding box that coincide with these precise annotation. These sequences are the ground truth. These have characteristics that highlight a specific or a combination of challenges like partial or full occlusion of objects, camera movements, illumination and scale change.

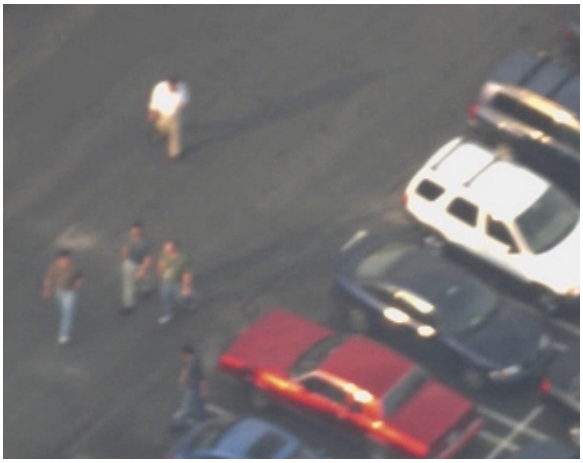
We shall look at some of these video sequence before using them to evaluate our algorithm. Figure 4.7 displays the selected sequences. The jumping sequence contains 313 frames of a person skipping ropes. The camera is not stable and with the target jumping there is a lot of blur. The car chase sequence has 9928 frames and has occurrences that cover multiple challenges like occlusion, distortion and identical cars along with the target. This sequence also has characteristics of being shot from an unstable camera as the video is taken on-board a helicopter in pursuit. The Pedestrian-3 is a small sequence having only 184 frames it gives us a birds eye view of pedestrians walking about a parking lot. Due to the close proximity of pedestrians and low resolution distinguishing their appearances is challenging. The last selected sequence from literature is David with 761 frames. This sequence highlights different light intensities, where the object-of-interest is a person walking through rooms with different light settings. The fastboard is one of our added sequence. It has 265 frames with the target object moving in and out of the cameras view.

4.4.1 Evaluation Protocol

The tracking algorithm to be evaluated is run on a video sequence. The results obtained are assessed against respective ground truth. The assessment is based on degree of overlap between the bounding box generated by the tracker and the annotated ground truth. Based on the overlap five possible cases can be inferred. If the degree of overlap is greater than a set threshold, it is considered to be a **true**

(a) Jumping

(b) Car chase



(c) Pedestrian-3

(d) Motocross

(e) David

(f) Fastboard

Figure 4.7: Video sequences selected to evaluate and compare related tracking algorithms

positive(TP). If the overlap is lower than the threshold, the bounding box counts as a **false positive**(FP) and the ground truth annotation counts as a **false negative**(FN). In case of an absent bounding box, when the ground truth annotation is present for a frame. The annotation counts for a false negative. While the opposite case presents a false positive. For a case of **true negatives**(TN), both annotation and the bounding box should not exist.

Using these terms, three performance metrics are measured to evaluate the tracking system. These are precision(P), recall(R) and F-Measure(F) [30]. Precision is formalized as, of all the classified positive examples how many are truly positive. Equation 4.1 states the measure.

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

Recall is formalized as, the proportion of truly positive examples that were correctly labeled to be positive. Equation 4.2 states the measure.

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

A system with high precision but with low recall, is too stringent, meaning a lot positive examples will be left unidentified. Hence, it's not reliable for long-term tracking.

A system that has high recall but with low precision, is too liberal with its classification. Meaning the results might include a lot of false positives. Hence, not reliable either.

There needs to be a trade-off between these two measures to provide the required results. Ergo we compute another metric called the F-measure to provide the harmonic mean of precision and recall. Equation 4.3 states the measure.

$$F = \frac{2PR}{P + R} \tag{4.3}$$

In the following section we present the analysis of the results from comparing SIFT infused TLD tracker against the traditional TLD, based on these performance metrics.

4.4.2 Analysis

In this section we present an evaluation of the results from comparing TLD modified with SIFT features against the traditional TLD on the selected video sequences. The purpose of introducing SIFT features to TLD was to reduce the number of tracking failure that occurred, when the target object reappears after moving out of the drones field of view immediately after initialization of the algorithm. The initialization stage is most vulnerable to failure as the system heavily relies on learning target representations to be able to perform tracking. Applications where both the target object and the camera source are in constant motion is critical for the tracker to obtain as may robust representations of the target as possible. We shall now look at the results from our experiments on evaluating the tracker on video sequences with challenging scenarios.

Table 4.1 provides details on number of true positive detections of the target object made by the traditional TLD tracker and our TLD_SIFT tracker. The column frames, reports on the number of frames in that video sequence. The column occlusions refer

Figure 4.9: Analysis procedure based on degree of overlap between bounding box and ground truth annotation. The red contour in the image is the manually annotated box referred to as ground truth. The green contour in the image is the bounding box generated by the algorithm. The overlap between them is represented by the green opaque box

to the number of frames in which the target is obstructed from view by another foreground object or the target is simply absent. The other two columns report the number of true detection made by the respective algorithms. Three sequences have distinguishing results on analysis. Each of these video sequences have a fast moving target object with non-fixed camera source. As shown in Figure 4.9, for the tracker detection to be a true positive detection, it has to have a high degree of overlap with the annotated box of that frame. The green box is the tracker generated bounding box while the red box is the manually annotated box over the target object.

The modified TLD algorithm has results on par with that of original TLD on video sequences Jumping, Pedestrian and David. In these the target object is either not moving rapidly or not many occlusions occur. Once the tracker learns a good amount of target representations of different appearances. The tracker is strong against fast moving targets and occlusion. In video sequence car chase and motocross there are multiple occlusions that occur as the target is moving fast. At few of the initial occlusion the target has some change in appearance when it returns within cameras

Sequence	**Frames**	**Occlusions**	**TLD**	**TLD_SIFT**
Jumping	313	0	313	313
Care Chase	9928	1268	5968	**5979**
Pedestrian-3	184	24	156	156
Motocross	2665	1253	1081	**1088**
David	761	0	761	761
Fastboard	259	140	11	**26**

Table 4.1: Illustrates number of true positives detected by the tracking algorithm. Incorporating SIFT features didn't provide much improvements when the object remains within the UAVs field of view. The fastboard sequence has the distinguishing results. This is because SIFT was incorporated primarily to handle fast moving object that can go in and out of the field of view immediately after initialization

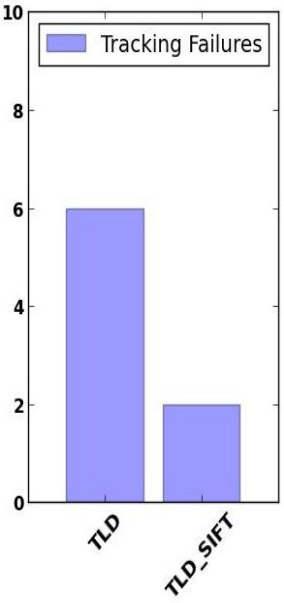


Figure 4.8: Average number of tracker failures on fastboard video sequence

field of view. It's on these frames that there are loss in target detection in the original TLD.

In the video sequence Fastboard, there are multiple instances where the target moves in and out of the cameras field of view. The first occlusion occurs 4 frames after initialization and the target reappearing has a slight change in its appearance. In such instances tracking systems have trouble re-initiating the tracker as the appearance of the target is new and not learned before. It is also evident from the results that TLD with SIFT features is relatively better at handling such instances. As even a few good features that remain resilient to modest changes in the target appearance is enough to re-initiate the tracker on the target object, which can then learn the new appearances.

The histogram on Figure 4.8 shows the average number of tracker failures that occurs while evaluating the fastboard video sequence for 10 runs. Here the tracking failure refers to the event when the tracking system is unable to detect the target on

	TLD			**TLD_SIFT**		
Sequence	**Precision**	**Recall**	**F-Measure**	**Precision**	**Recall**	**F-Measure**
Jumping	100	100	100	100	100	100
Car Chase	85.21	68.91	76.20	**85.34**	**69.04**	**76.33**
Pedestrian-3	100	98.73	99.36	100	98.73	99.36
Motocross	76.56	88.97	82.30	**77.22**	**89.55**	**83**
David	100	100	100	100	100	100
Fastboard	61.11	7.85	18	**81.25**	**18.57**	**30.23**

Table 4.2: Illustrates Precision, recall and F-measures of the two algorithms evaluated on the selected video sequences. Results from TLD_SIFT shows no deterioration in tracking compared to the traditional TLD algorithm. But doesn't provide much improvements when the target object's movements remain within the field of view. Results from motocross and fastboard both show increased stability in scenarios where source camera and the target are at constant motion with occlusion at initial stages of the algorithm.

re-entry for more than 5 sec. Further, Table 4.2 reports the precision, recall and the F-measure calculated for the two algorithms evaluating the selected video sequences. The values in bold indicate improved results.

This modification reinforces the algorithm to perform better in scenarios where the target moves in and out of the drones field-of-view immediately after initialization, when there is inadequate amount of training examples of the target object. Which is often the case with an UAV, as the algorithm does take a second to initialize and a constantly moving object's slight change in appearances can easily cause the tracker to fail and consequently making the drone incapable of following the target. Each true target detection is crucial in providing the feedback to steer the drone towards the target and to avoid tracking failure. Chapter 5 concludes this thesis and discusses possible avenues of future work.

Chapter 5

Conclusion

This thesis implemented and demonstrated a novel real-time tracking and following paradigm for UAVs and UGVs. Constructed by combining adaptive long-term tracking system and an effective following mechanism. The communication was made seamless by implementing the system on ROS framework. Applied on an UAV, it was able to maneuver autonomously while tracking and following a target, that was initialized during runtime. The UAV was able to perform swift and smooth transitions. This was achieved by addressing challenges on both the tracking and the following mechanism. Primarily, the perception of depth. Acknowledging that, an efficient following mechanism that retains the target within UAVs field of view was crucial as the tracking is only effective as long as the target is visible to the tracking system. We exploited UAV's on-board stereo cameras to generate a depth map of the local scene to augment the tracking and the following mechanism with valuable depth information. This facilitated in UAV's swift response to target movements and to maintain a set distance to the target at all time. The depth was also used by the following mechanism to avoid collisions.

In an environment where both the source camera and the target are not stationary. It was also important to recognize the volatility of the trackers initialization process. Being an adaptive on-line learning system it was crucial for the tracking system to detect and track more resilient object features to cope with small appearance changes. We achieved this by modifying the tracking system to introduce more resilient SIFT features to work with. As evaluated on video sequences, this lowered tracker failures.

There are still several challenges that are to be addressed to minimize tracking failures. The most difficult instance is again immediately after initialization. Our work can only cope with modest appearance changes, when there is considerable change to the target's appearance, tracking is no longer possible leading to immediate

stoppage. This challenge has to be mitigated without defeating the purpose of on-line adaptive learning. Another challenge is coping with growing templates of the targets appearances. Though every instance of the target within drones vision gives the tracking system more templates with valuable information to work with. It also increases computational cost affecting the overall performance, as the tracking goes on. One temporary approach to coping with this problem is to stop learning when the number of stored training examples start to cause a drop in the frame rate of the tracking algorithm. That is, the number of frames the tracking algorithm can process in a second. With that said, the complications occurs when there is considerable change in targets appearance after halting the learning process. This could again lead to immediate stoppage.

Parallax on the stereo camera provides means to generate depth, as such the scene visible to the tracking system was only from the left camera. The drone's field of view can be extended by efficiently switching between the cameras, to provide more stability to the tracking system.

In the event of loosing the target object. Instead of stopping the drone, a target recovery mechanism can be incorporated that can initialize autonomously on loosing the target. Such a mechanism would involve performing search by visually scanning its environment based on path planning.

Solving such problems would drastically improve systems performance with minimal failures. Which is imperative for any real-time application. Our work done in this thesis is a step directed towards generating more such adaptive autonomous vehicular tracking and following systems.

Bibliography

[1] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision*, 56(3):221–255, 2004.

[2] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM, 1998.

[3] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[4] Roberto Brunelli. Template matching techniques in computer vision, 2008.

[5] Robert T Collins, Yanxi Liu, and Marius Leordeanu. Online selection of discriminative tracking features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1631–1643, 2005.

[6] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Real-time tracking of non-rigid objects using mean shift. In *IEEE Conference on Computer Vision and Pattern Recognition, 2000. Proceedings.*, volume 2, pages 142–149. IEEE, 2000.

[7] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):564–577, 2003.

[8] Paul Fahlstrom and Thomas Gleason. *Introduction to UAV systems*. John Wiley & Sons, 2012.

[9] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[10] Xiang Gao, Terrance E Boult, Frans Coetzee, and Visvanathan Ramesh. Error analysis of background adaption. In *Proceedings. IEEE Conference on Computer Vision and Pattern Recognition, 2000.*, volume 1, pages 503–510. IEEE, 2000.

[11] Ismail Haritaoglu, David Harwood, and Larry S Davis. W 4: Real-time surveillance of people and their activities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):809–830, 2000.

[12] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Proceedings of the Alvey vision conference*, volume 15, page 50. Citeseer, 1988.

[13] Ramesh Jain and H-H Nagel. On the analysis of accumulative difference pictures from image sequences of real world scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2):206–214, 1979.

[14] Allan D Jepson, David J Fleet, and Thomas F El-Maraghi. Robust online appearance models for visual tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1296–1311, 2003.

[15] Zdenek Kalal, Jiri Matas, and Krystian Mikolajczyk. Pn learning: Bootstrapping binary classifiers by structural constraints. In *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 49–56. IEEE, 2010.

[16] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Face-tld: Tracking-learning-detection applied to faces. In *17th IEEE International Conference on Image Processing (ICIP)*, pages 3789–3792. IEEE, 2010.

[17] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In *20th International Conference on Pattern Recognition (ICPR)*, pages 2756–2759. IEEE, 2010.

[18] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1409–1422, 2012.

[19] Tomáš Krajník, Vojtěch Vonásek, Daniel Fišer, and Jan Faigl. Ar-drone as a platform for robotic research and education. In *Research and Education in Robotics-EUROBOT 2011*, pages 172–186. Springer, 2011.

[20] James Kramer and Matthias Scheutz. Development environments for autonomous mobile robots: A survey. *Autonomous Robots*, 22(2):101–132, 2007.

[21] JP Lewis. Fast normalized cross-correlation. In *Vision interface*, volume 10, pages 120–123, 1995.

[22] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[23] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[24] Michael Margolis. *Arduino cookbook*. " (1st ed. O'Reilly Media, Inc. Sebastopol", 2011.

[25] Johannes Meyer, Alexander Sendobry, Stefan Kohlbrecher, Uwe Klingauf, and Oskar von Stryk. Comprehensive simulation of quadrotor uavs using ros and gazebo. In *Simulation, Modeling, and Programming for Autonomous Robots*, volume 7628, pages 400–411. Springer, 2012.

[26] Hieu T Nguyen, Marcel Worring, and Rein Van Den Boomgaard. Occlusion robust adaptive template tracking. In *Eighth IEEE International Conference Proceedings on Computer Vision, 2001. ICCV*, volume 1, pages 678–683. IEEE, 2001.

[27] Kenzo Nonami, Farid Kendoul, Satoshi Suzuki, Wei Wang, and Daisuke Nakazawa. *Autonomous Flying Robots: Unmanned Aerial Vehicles and Micro Aerial Vehicles*. Springer Science & Business Media, 2010.

[28] Mustafa Ozuysal, Pascal Fua, and Vincent Lepetit. Fast keypoint recognition in ten lines of code. In *IEEE Conference on Computer Vision and Pattern Recognition, 2007. CVPR'07*, pages 1–8. IEEE, 2007.

[29] Jeffrey E Passner, Robert E Dumais Jr, Robert Flanigan, and Stephen Kirby. Using the advanced research version of the weather research and forecast model in support of scaneagle unmanned aircraft system test flights. Technical report, DTIC Document, ARL-TR-4575, U.S. Army Research Laboratory: White Sands Missile Range, 2009.

[30] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning, TR SIE-07-001*, (2):37–63, 2011.

[31] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[32] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China*, pages 1–4. IEEE, 2011.

[33] Haim Schweitzer, JW Bell, and Feng Wu. Very fast template matching. In *Computer Vision ECCV 2002*, pages 358–372. Springer, 2002.

[34] Stephen Se, David Lowe, and Jim Little. Vision-based mobile robot localization and mapping using scale-invariant features. In *IEEE International Conference on Robotics and Automation, 2001. Proceedings 2001 ICRA.*, volume 2, pages 2051–2058. IEEE, 2001.

[35] Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94.*, pages 593–600. IEEE, 1994.

[36] Tal Shima and Steven J Rasmussen. *UAV cooperative decision and control: challenges and practical approaches*, volume 18. SIAM, 2009.

[37] Utkarsk Sinha. Scale invariant feature transform. `http://www.aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/`, 2010.

[38] Russell Smith et al. Open dynamics engine. `http://ode.org/`, 2005.

[39] Jan Erik Solem. *Programming Computer Vision with Python: Tools and algorithms for analyzing images.* " O'Reilly Media, Inc.", 2012.

[40] Chris Stauffer and W Eric L Grimson. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):747–757, 2000.

[41] Kah-Kay Sung and Tomaso Poggio. Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):39–51, 1998.

[42] Sukeshini N Tamgade and Vibha R Bora. Motion vector estimation of video image by pyramidal implementation of lucas kanade optical flow. In *2009 2nd International Conference on Emerging Trends in Engineering and Technology (ICETET)*, pages 914–917. IEEE, 2009.

[43] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2001. CVPR 2001.*, volume 1, pages I–511. IEEE, 2001.

[44] Paul Viola, Michael J Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. In *Ninth IEEE International Conference on Computer Vision, 2003. Proceedings.*, pages 734–741. IEEE, 2003.

[45] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *Acm computing surveys (CSUR)*, 38(4):13, 2006.

[46] Qian Yu, Thang Ba Dinh, and Gérard Medioni. Online tracking and reacquisition using co-trained generative and discriminative trackers. In *Computer Vision–ECCV 2008*, pages 678–691. Springer, 2008.