Dynamic Object Detection Using Moving Cameras for UAV Collision Prevention

By

Arman Dadrass

A Project Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering

in the Department of Electrical and Computer Engineering

Supervisory Committee

Dr. Amirali Baniasadi
Department of Electrical and Computer Engineering)

Dr. Lin Cai
Departmental of Electrical and Computer Engineering

Dr. Sardar Malek
Department of Electrical and Computer Engineering

ABSTRACT

In this project, the focus is on object and motion detection in video streams from a camera mounted on a moving platform, such as a drone. Motion segmentation and moving object detection are fundamental for object tracking and collision avoidance in autonomous vehicles and flying drones. Moving object detection and tracking with a moving camera is challenging due to the combined effects of the camera's motion relative to its mounting base and the movement of the platform within the environment. The limited field of view of the camera results in frequent object discontinuity and severe background variations, which conventional background subtraction approaches for fixed cameras cannot handle. To address these challenges, dense optical flow clustering is used to detect moving objects with a moving camera. The clusters correspond one-to-one with moving objects and background motion due to camera's motion; however, objects frequently enter or leave the scene, necessitating frequent redefinition and recalculation of clusters. Additionally, since the number of objects in the scene is unknown and can vary over time, the clustering algorithm must adapt quickly to changing scenarios. Therefore, the Adaptive Resonance Theory-2 (ART2) network was adapted to eliminate the need for pre-tuning the number of clusters as a hyper-parameter, automating the process similarly to human perception and enabling rapid redefinition and recalculation of varying cluster numbers. The performance of the proposed approach was evaluated in terms of execution time and accuracy using the VisDrone and KITTI datasets, and the results are discussed.

DEDICATION

Just hoping this is useful!

Contents

Chapter 1

Introduction

1.1 Background

Computer vision and artificial intelligence advancements have evolved significantly over the past few decades. Several enhancements have resulted in positive outcomes across a wide range of applications, especially for autonomous driving or robotics and security systems. In those applications, one of the key functions is detecting and tracking moving objects in dynamic and often unpredictable environments.

The detection of moving objects is an essential step that has to be performed in many applications, such as Obstacle Avoidance, Path Planning, and Human-Robot Interaction. The ability to perceive and react to moving objects in dynamic environments is a key requirement for robots that are expected to operate autonomously in factories, homes or public spaces.

The object detection problem in video streams has long been studied in the literature. The problem has been solved, considering the cases with fixed cameras against a background somewhere [1,2], and now the problem is being regarded as a trivial task in image and video processing within such a setup. The basic idea is to subtract the current frame of the stream from a background model to figure out the changes and isolate them with respect to the stationary background, which leads to a foreground mask. Figure 1.1 represents the simple block diagram of this process.

The main component of these algorithms is background modelling. Following that, the remaining part is subtracting the current frame from the background model and then applying a threshold for de-noising purposes to generate the foreground mask. It is not practical to use the same background model all the time since there

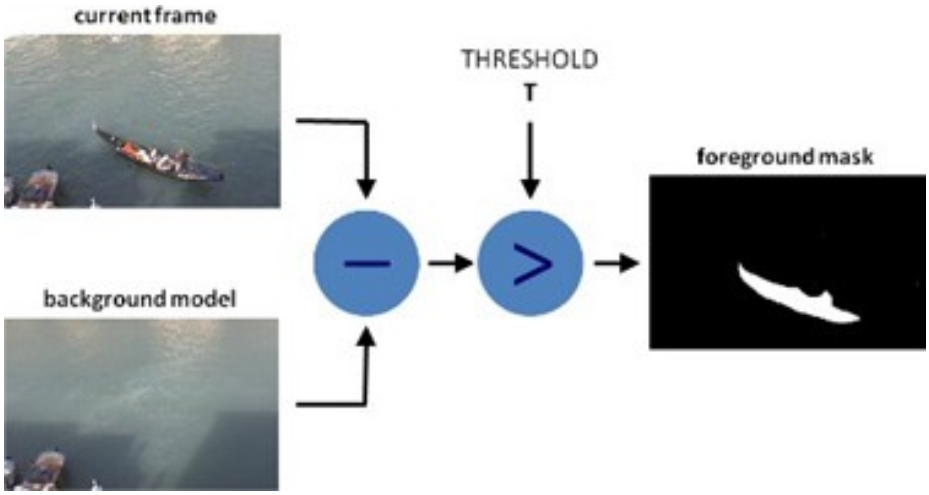


Figure 1.1: Simple Background subtraction algorithm [3]

are always some very slow changes in the background, and also, during the day or due to other changes, the illumination will vary. Therefore, we should update the background model regularly to take into account all these slow variations (compared with the video frame rate) in the background to have a robust algorithm.

One of the most commonly used background modelling schemes is the Mixture of the Gaussian model (MOG) [4], in which a mixture of k-Gaussian distribution models the spatial probability distribution of the pixels. Usually, k is chosen to be 3, 4, or 5, as the maximum number of the Gaussian distributions and the covariance matrices are considered to be diagonal, often for computational purposes. This algorithm has other extensions; for example, instead of using simple first-order difference equations to update the weights, we can use higher-order dynamics to take into account previous frames as well. Besides, to improve spatial robustness, instead of pixel-by-pixel calculations, further details regarding the spatial dependency of the pixels can be considered. Efficient implementations of these algorithms are available in The OpenCV library for video processing under the names MOG and MOG2, in which other rubustifications have also been considered.

With the rise of drones and autonomous vehicles, mounted cameras are regarded as one of the major role players in these sensory systems. Studies on moving cameras have emerged over time. In a stationary camera, static objects are static scenes in the streams, and the images project the 2D motion structure of the moving objects. In contrast, in a moving camera, everything appears moving because of the camera's motion.

When the camera starts to move, the background subtraction algorithms with slow adaptation lose ground since the background is no longer stationary or almost stationary. The scene will change as a result of camera motion, and therefore, even the background objects appear to be moving. Consequently, the variations from one frame to the other in the weights of the clusters would be high, and the principal clusters would become indistinguishable. In other words, the result of foreground detection would become highly dependent on the background representation.

1.2 Objectives of the Study

Each video frame represents a projection of the environment's illuminates and intensity onto a finite image plane, and assuming that the frame rate is significantly higher than the motion dynamics—typical in most scenarios—the primary change between consecutive frames is the displacement of pixel values and feature points. Consequently, tracking these pixels and feature points across frames can provide valuable insights into the motion structure within video streams. This principle underpins the concept of optical flows, which measure the displacement of pixels and feature points between frames. Optical flows have proven practical for analyzing the motion structure projected onto the image plane, enabling more effective understanding and investigation of the dynamics present in video sequences.

Although optical flow patterns are usually noise-corrupted and do not exactly capture the motion fields due to complexities in their evaluation, they usually reveal the motion field patterns and such patterns can be used for separating different types of motions being encoded in consecutive frames. Unfortunately, encoding the possible patterns themselves is a very challenging problem, and therefore, in analyzing optical flows for revealing motions, mostly data-driven feature selection algorithms have been used. This highlights the need for Efficient pattern recognition algorithms to be used for revealing the structure of motion from the optical flows, which enables considering the above-mentioned complexities

The Adaptive Resonance Theory 2 (ART2) network is a type of neural network that is particularly suited for pattern recognition and clustering in noisy environments. Here is how it can help in moving object detection with a moving camera while being used along with the optical flow maps:

1. **Adaptive Learning**

- **Stable and Plastic Memory:** ART2 networks can learn new patterns without forgetting previously learned ones, making them suitable for environments where the background and object appearance change over time.

2. **Noise Robustness**

   - **Handling Noise:** ART2 networks are designed to handle noisy input data, which is common in scenarios with moving cameras where motion blur and dynamic backgrounds introduce significant noise.

3. **Real-Time Adaptation**

   - **Continuous Learning:** The network can continuously adapt to new patterns in real-time, which is crucial for processing video frames on the fly as the camera moves.

4. **Pattern Recognition**

   - **Clustering Capabilities:** ART2 excels in clustering similar patterns, helping to group similar motion vectors and distinguish between background and moving objects despite the complex motion patterns induced by the moving camera.

The above-mentioned characteristics of the ART2 network make it a particularly suited approach for extracting the structure of motion from optical flow maps, especially for real-time applications, which will be investigated in this project. The rest of this report is organized as follows: In the next chapter, related works in moving object detection with moving cameras will be briefly reviewed. Chapter three will review the data sets being used in this work and the implementation setup. Chapter four will detail the proposed method and the mathematical behind the curtain. Lastly, the experimental results and related discussions, along with future directions, will be provided.

1.3 Challenges in Moving Object Detection with Moving Cameras

Authors in [5] provide a comprehensive survey of the works and techniques for moving object detection with moving cameras. Moving object detection with a moving camera

is challenging due to several factors as classified by [5]:

1. Camera Motion
    - Complex Motion Patterns: Moving cameras often follow complex paths, making it diffi- cult to distinguish between background motion and actual object motion.
    - Motion Blur: Rapid camera movements can cause motion blur, complicating the detection of objects.
2. Background Dynamics:
    - Changing Background: The background in the scene is continuously changing, which makes it hard to establish a consistent reference for detecting moving objects.
    - Parallax Effect: Objects at different distances from the camera move at different relative speeds, adding complexity to the motion patterns observed.
3. Object Appearance Changes:
    - Scale Variation: Objects appear at different scales as they move closer to or farther from the camera.
    - Occlusions: Moving objects can become occluded by other objects or by the edges of the camera's field of view, making consistent tracking difficult.
4. Lighting Conditions:
    - Variable Lighting: Changes in lighting, such as shadows, reflections, or varying ambient light, can affect the appearance of objects and the background, where such variations would be more severe as the result of the camera and mounting base motion.

The challenges mentioned above are mainly caused by several root factors: the intricate motion dynamics of both the camera and its mounting base in relation to the environment, the diverse characteristics of objects and their movements, the difficulty of projecting the environment and moving objects onto a limited image plane, variations in perceived intensities due to changes over time and motion, and

the inherent complexity of highly unstructured environments, where predicting a range of possible events is difficult.

Chapter-II: Related Works

This chapter provides a brief literature review on the use of clustering techniques for moving object detection with moving cameras, specifically focusing on the application of the ART2 neural network in the analysis of optical flow maps.

Detecting moving objects in video sequences captured by moving cameras is a challenging problem, particularly in complex environments with dynamic backgrounds. Various approaches, ranging from traditional analytical methods to advanced deep-learning techniques, have been proposed to tackle this issue. A comprehensive survey of these techniques is provided in [5], where the authors systematically review the methods developed for moving object detection in such scenarios. Another valuable survey is presented in [6], which categorizes the proposed approaches into four primary groups: modeling-based background subtraction, trajectory classification, low-rank and sparse matrix decomposition, and object tracking. The authors offer a detailed analysis of the underlying principles of each category, highlighting the strengths and limitations of the various methods.

In [7], an analytical approach is introduced for detecting moving objects using a background orientation field reconstruction technique based on Poisson fusion. The method reconstructs the background orientation from the modified gradient of the optical flow orientation, which remains unaffected by scene depth. This orientation field serves as a spatially continuous variable that encapsulates both global and local motion information. The comparison between the original and reconstructed orientation fields yields a motion saliency map. To enhance the saliency of moving objects, a weighted accumulation method is proposed, which also improves the consistency between object and background regions. Furthermore, the approach integrates motion continuity to reduce false positives in the motion saliency map.

The study in [8] addresses the problem of detecting moving objects in video frames captured by a moving camera by first computing dense optical flows and then applying the RANSAC algorithm to calculate a homography matrix. This matrix is used to generate background optical flow vectors, and the motion saliency is derived from the differences in magnitudes and the cosine of the angle between the original and background optical flows. The difference in magnitudes is particularly effective in the absence of significant zooming effects, while the angle measure proves useful when zooming is present.

In [9], the authors introduce a framework known as optical flow templates, which captures detailed information about environmental structure from instantaneous image motion in a camera mounted on a mobile robot. These templates encode possible flow fields due to ego-motion, depending on the environment's shape and the robot's attitude. By labeling superpixels with these templates, the method can differentiate between traversable space and obstacles. The framework is particularly effective in urban driving scenarios, where it demonstrates both qualitative and quantitative success in identifying ground planes and obstacles, such as vehicles and lampposts. One of the notable advantages of this approach is its low computational complexity, achieving per-frame computation times of 20 ms, excluding the time required for optical flow and superpixel calculations.

The work presented in [10] introduces MatchFlow, a novel approach to optical flow estimation that improves performance by using Geometric Image Matching (GIM) as a pre-training task. MatchFlow leverages GIM to enhance feature extraction, benefiting from extensive real-world data. The method employs a QuadTree attention-based network, pre-trained on MegaDepth, to refine features for accurate flow regression. MatchFlow shows significant improvements in cross-dataset performance, achieving leading results in optical flow estimation while maintaining efficient computation and memory usage.

In [11], the authors propose a training-free method for moving object detection using a novel and efficient clustering algorithm. This method measures dense optical flow between consecutive frames, applies clustering to the histogram of optical flow orientations to segment nearby moving objects, and verifies the consistency of motion vectors for each candidate object. Experimental results on three

public datasets demonstrate that this method achieves significantly faster processing speeds (8.01 frames per second compared to 1.25) and higher recall (87.2% compared to 83.5%), with a precision of 93.5% compared to 89.8%, surpassing state-of-the-art algorithms.

The application of the ART2 neural network in clustering tasks is addressed in several studies. In [12], the authors propose a modified version of the ART2 neural network that addresses limitations in the classical ART2 model, particularly in how similarity between patterns is measured. The new approach introduces Euler distance as a similarity measure and retains the magnitude of input patterns throughout the processing stages, including pattern matching and vigilance testing. This modified ART2 classifier, enhanced with three special functions for better similarity measurement, outperforms the classical ART2 model in clustering tasks, as demonstrated by experimental results.

Further advancements in ART2 networks are discussed in [13], where the authors introduce an enhanced version of the ART2 network, known as ETM-ART2. This version addresses the instability in clustering results that often occurs in the classic ART2 model, where categories may change with slight variations in training conditions. The ETM-ART2 model incorporates an improved triplex matching mechanism to overcome these issues, and experimental results show that it outperforms the classic ART2 in various clustering tasks, making it a more effective and versatile algorithm for data mining applications.

The work in [14] focuses on the Adaptive Resonance Theory under Constraint (ART-C 2A), an enhanced learning paradigm based on ART 2A. ART-C 2A is designed to generate a user-defined number of recognition nodes by dynamically estimating an appropriate vigilance threshold. Empirical experiments compare ART-C 2A with ART 2A, online K-Means, batch K-Means, and SOM, evaluating cluster validity and learning efficiency. ART-C 2A retains the online cluster creation capability of ART 2A while offering an alternative solution that allows direct control over the number of output clusters produced by the self-organizing process.

An application of the ART2 network in computer vision is presented in [15], where the authors propose a new neural network architecture based on ART2, specifically designed for color image segmentation. This new ART2 model introduces a similarity measurement mechanism that considers both the magnitude and phase of input vectors, incorporating spatial information in the feature space. The enhanced architecture retains the classical ART2 features, such as self-organizing learning and the ability to categorize without a predefined number of clusters. It also improves clustering performance and addresses the pattern-shifting problem seen in the classical ART2. Experimental results show that this new ART2 architecture effectively performs unsupervised segmentation in a modified Luv color space, accurately measuring perceptual color differences using spatial information.

Chapter-III: Datasets and Implementation Setup

The ultimate objective of the proposed algorithm is to detect moving objects through drone-mounted cameras for both object tracking and collision avoidance. This guides us to two available datasets prepared for this purpose.

VisDrone

VisDrone [16] is a comprehensive benchmark featuring meticulously annotated ground truth data for key computer vision tasks aimed at integrating vision technology with drones. The VisDrone2019 dataset, created by the AISKYEYE team at the Lab of Machine Learning and Data Mining, Tianjin University, China, includes 288 video clips made up of 261,908 frames and 10,209 static images. These were captured by various drone-mounted cameras across diverse settings, encompassing 14 cities in China, both urban and rural environments, and a variety of objects such as pedestrians, vehicles, and bicycles. The scenes vary in density from sparse to crowded. The dataset includes footage from different drone models and scenarios with various weather and lighting conditions. Over 2.6 million bounding boxes are manually annotated, identifying common targets like pedestrians, cars, bicycles, and tricycles. Additional attributes such as scene visibility, object class, and occlusion are also provided to enhance data utility.

The dataset consists of five categories: Object Detection in Images, Object Detection in Videos, Single-Object Tracking, Multi-Object Tracking, and Crowd Counting. For this project, we focused specifically on the Multi-Object Tracking portion, which includes seven sequences of video recordings captured by drone-mounted cameras. These sequences are used for detecting and tracking moving objects. The primary reason for selecting the VisDrone dataset for testing and verification is its ability to simulate a realistic environment for our ultimate goal—object tracking with patrolling drones. Since the dataset is composed of videos recorded from drone-mounted cameras, the resulting camera motion closely mimics what we might encounter in real-world scenarios

Figure 2 showcases multiple consecutive frames of an example taken from the Multi-Object Tracking dataset of VisDrone.

(a) Frame 1 (b) Frame 13 (c) Frame 25

Figure 2: Multiple frames of single video in VisDrone Multi-object Tracking dataset

KITTY dataset

The VisDrone dataset is mainly focused on object tracking using drones. In such a scenario, usually, the drone is moving at higher altitudes and aiming to detect and track moving objects on the ground. In these scenarios, the variations within the field of view slowly change, and therefore, detecting moving objects would be different compared with scenarios in which drones are moving in cluttered environments and need to avoid moving objects to prevent a collision. VisDrone dataset is not a good option for evaluating and verifying moving object detection and tracking for this latter scenario, and unfortunately, there is no verified data set for such a purpose. However, there are datasets such as KITTI [17] provided for cameras mounted on ground vehicles, which can also be

used for this purpose, considering the limitations in emulating the possible drone motions and the resulting camera motions.

The providers utilized their autonomous driving platform, Annieway, to create innovative and challenging real-world computer vision benchmarks for evaluating algorithms for stereo vision, optical flow, visual odometry, 3D object detection, and 3D tracking. To gather the necessary data, they have equipped a standard station wagon with high-resolution color and grayscale video cameras. Accurate ground truth is provided by a Velodyne laser scanner and a GPS localization system. Their datasets are collected from various environments, including the mid-size city of Karlsruhe, rural areas, and highways, with up to 15 cars and 30 pedestrians visible per image. They offer all data in its raw format and have extracted specific benchmarks for each task, along with evaluation metrics and an evaluation website.

For the purpose of testing and verification in this project, in particular, we have used the flow 2015 dataset of the KITTI, which includes 20 sequences of videos recorded by the car-mounted camera, for purpose of detecting moving objects and generating optical flows.

The following figure 3 showcases multiple consecutive frames of a video from a KIITI dataset for multi-object tracking.

(a) Frame 44 (b) Frame 0 (c) Frame 62

Figure 3: Multiple frames of single video in KITTI Multi-object Tracking dataset

Implementation Setup and Code Pipeline

For the implementation and testing of our video processing algorithms, we utilized a personal laptop configured with an Intel 7th generation Core i7 CPU, 16GB of RAM, and an Nvidia GeForce GTX 1050 4GB Pascal GPU. The software environment included OpenCV, built with Intel-specific accelerations, CUDA GPU acceleration, and the Intel Python distribution.

Intel acceleration kernels, which are not free for commercial use, were employed solely for research purposes in this project. CUDA technology was extensively used for computationally demanding tasks, such as dense optical flow calculations. In cases where CUDA implementations were unavailable, we improved performance by utilizing OpenCL and the UMat data structure, which offers advantages over the Mat data structure.

To optimize code execution, we applied memory pre-allocation and asynchronous stream assignment. This approach allows the CPU to initiate commands and return to the main thread without waiting for completion. We used a minimum of two GPU streams to ensure that the CPU could start a second stream if the first completed faster than the GPU. This method ensures efficient processing by allowing the CPU to manage tasks effectively and wait for all assigned streams to complete before repeating the process.

In addition to the OpenCV CUDA module, we explored Python packages such as Numba and TensorFlow for writing CUDA code. TensorFlow's newer versions provide enhanced Python integration and GPU optimization capabilities. However, managing GPU usage can be challenging due to TensorFlow's full GPU capacity claim upon invocation.

We also considered Intel-specific optimizations and the Intel Python distribution to boost performance. Another acceleration method explored was Cython, which is particularly effective for image pixel loop operations. Cythonized implementations can lead to significant performance improvements even with standard Python distributions. Although a detailed discussion on Cython is outside

the scope of this report, numerous tutorials are available online that focus on image processing loops. Combining Cython with the Intel Python distribution can further enhance code performance.

Chapter-IV: Proposed Moving Object Detection Method

This chapter provides details of the implemented pipeline for processing video frames captured by a moving camera for moving object detection.

The system architecture for this project is based on two main threads: the video capture/read thread and the processing thread. The video capture thread initializes a video streaming class to acquire frames from various sources, including webcams, locally stored videos, or IP addresses specified by the user. This setup supports real-time processing, such as connecting a base station to a drone.

The capture thread manages the saving of video frames into a queue for subsequent processing. It handles data type conversions, initialization, and memory management. Additionally, it evaluates the algorithm's processing rate in comparison to the video capture frame rate, which is influenced by the capturing camera or the rate at which video is read from memory.

The processing thread is divided into four primary components: frame filtering and preprocessing, global motion estimation, optical flow calculation, and flow clustering for moving object detection and tracking. The following subsections elaborate on each component. Figure 4 describes the flow of the overall proposed pipeline for motion detection.

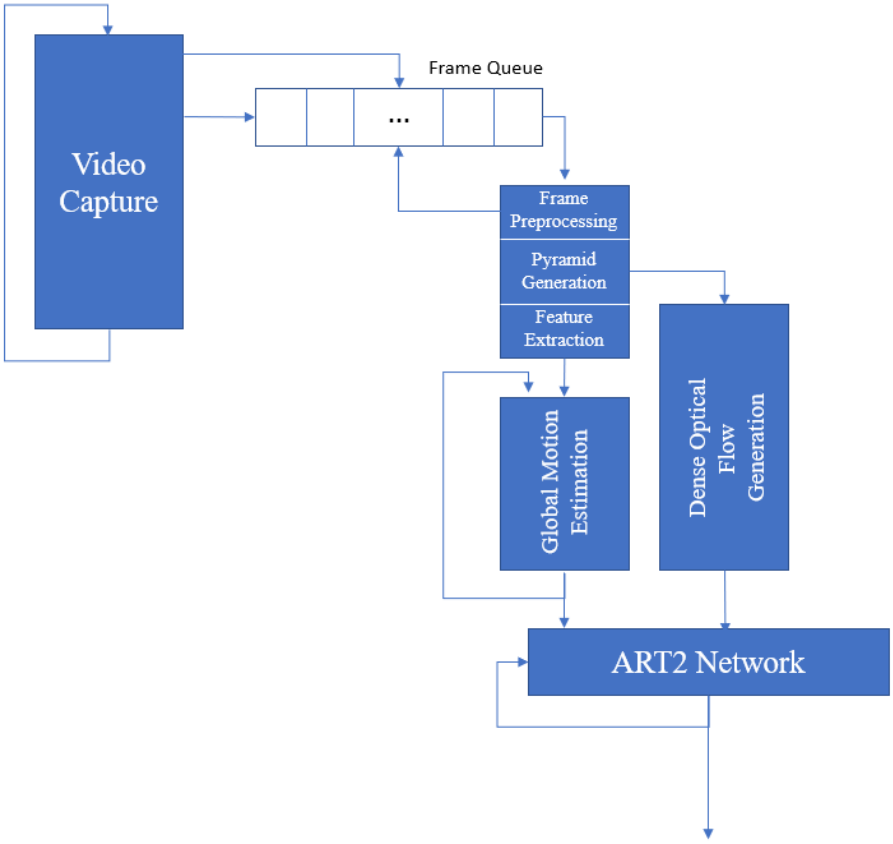


Figure 4: The proposed pipeline for moving object detection in moving camera streams.

Frame Preprocessing, Pyramid Generation, Feature Extraction

This module provides the remaining modules of the pipeline, multiple instances of the same frame undergoing various filtering processes. Multiple operations will be performed on each frame to enhance motion and optical flow estimation and the final clustering results. These operations are crucial for improving the accuracy and reliability of subsequent steps in the analysis pipeline.

Key preprocessing tasks in this module include:

- Preliminary Denoising: Frames captured directly from the camera's raw data streams often require initial denoising to reduce noise and improve quality. This step is essential for ensuring that subsequent processing stages are based on cleaner and more accurate data. In particular, here, we use a simple median filter to reduce the frame noise. In our code, the captured data will be read from the queue and will be updated into the Cuda kernel; the uploaded frame will then be filtered and will be passed to the next step, which is pyramid generation.
- Frame Pyramid Generation: Spatial filtering is employed to generate multi-resolution pyramids. This technique involves creating a series of progressively smoothed versions of the frame, which helps in capturing details at different scales and enhances the accuracy of motion and optical flow estimation. In particular, we have used Gaussian pyramids for investigating motion structures within different scales that will later be used for improving camera motion estimation and compensation.
- Extracting frame features: This step identified salient features of the reading frame, including Shi-Thomasi's good feature to track.
- Maximally Stable Extermal Regions (MSER): The MSER algorithm determines the salient object within each frame, which then can be used for improving the optical flow clustering results.

These operations collectively contribute to refining motion and optical flow estimation, leading to more accurate and robust results in the analysis of video frames and clustering motion information.

Frame Pyramid Generation [18]

Applying optical flows to the original image can be used in finding the structure of small motions compared with the image resolution; it fails in the case of large motions available. To overcome this problem, we can study the structure of motion for different resolutions, while higher resolutions reveal smaller motions and lower resolutions reveal larger ones. It should be noted that, although lower resolutions can be used to find the larger motions, small features will be lost in the frame, and their motions (even large motions) will be ignored. Therefore, there is always a limitation on the size of the features and the possible motions that can be revealed; in other words, the detection of large motions of small objects would always be harder.

An image pyramid is a collection of images generated from the original image by downsampling or upsampling the pixels up to a certain resolution. The procedure is depicted in figure 5 below:

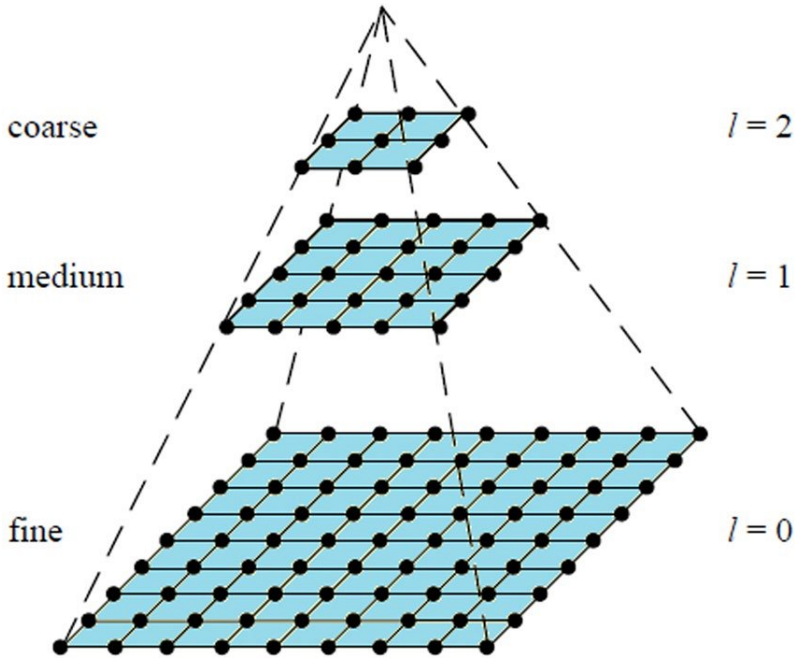


Figure 5: Different resolution level by downsampling an image pyramid [19]

Spatial filtering can implement downsampling and upsampling. This paper describes the rationale behind using Convolutional Artificial network architectures in machine learning-based optical flow algorithms such as RAFT.

Two well-known approaches for downsampling and upsampling images are the Gaussian and Laplacian image pyramid constructions. In the Gaussian pyramid algorithm, to downsample an image usually, we use the following procedure:

- Apply the following Gaussian kernel and convolve it with the image to filter the image:

$$K = \frac{1}{16}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \tag{1}$$

- Remove all the even rows and columns.

This way, we can reduce the resolution.

To increase the resolution, we will first add zero rows and columns between the current row and columns, then apply the previous Gaussian filter and multiply the result by 4.

The Laplacian pyramid will be constructed by first constructing the Gaussian pyramids and then finding the difference between the upsampled version of each pyramid layer starting from the lowest resolution with the corresponding layer in the Gaussian pyramid. In fact, the Laplacian pyramid describes the features that are available at each layer but not those that are available in the previous layer.

Shi-Tomasi Good features to track: [20–22]

The Shi-Tomasi algorithm is a variant of the Harris corner detection algorithm, which has conventionally been applied to find the corners in an image.

In the Harris corner detection algorithm, firstly, we consider a window composed of some pixels with a window function, which is usually a Gaussian or rectangular function (which is one over the window and zeros otherwise). The following figure summarizes different situations when sliding a window in an image:

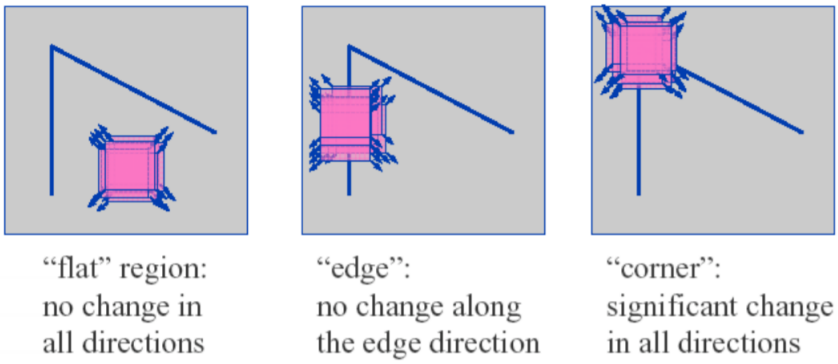


Figure 6: Harris corner detection idea [20]

By associating the window with the following window intensity function, we can mathematically formulate the problem. The notation I(x+u, y+v) represents the intensity value of a pixel in an image at a specific location after shifting it by (u, v).

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y,v) - I(x,y)]^2 \tag{2}$$

where $w(x,y)$ is the windowing function and $I(x,y)$as before stands for image intensity function.

First, we can simplify the window intensity function by substituting the Taylor expansion of the terms in the bracket as follows:

$$E(u,v) = \sum_{x,y} w(x,y)[I_x u + I_y v]^2 = \sum_{x,y} w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{3}$$

Define the matrix M as follow:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{4}$$

To separate the three mentioned situations for each window, we go through the following algorithm:

First, we will compute the gradients in both directions in the image, and then, using the windowing function and those gradients, we will find M. The window intensity function approximately is a quadratic function with the kernel matrix M based on the eigenvalues of this matrix the following three situations can occur:

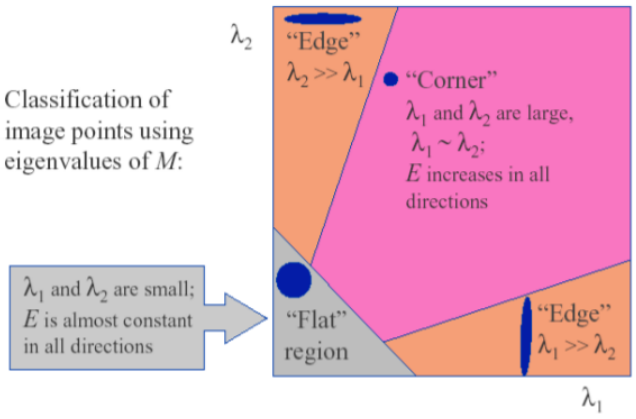


Figure 7: Different possible situation based on the eigenvalues of M [21]

Therefore, we can easily use the eigenvalues to find out if the window contains a corner or an edge and also which type of edge.

In the Harris algorithm, originally, instead of finding the eigenvalues, the following scoring function is used, which increases the execution speed, and therefore, there is no need to find the exact eigenvalues anymore:

$$R(\lambda_1, \lambda_2) = det(M) - k(trace(M))^2 \tag{5}$$

where from linear algebra we know that $det(M) = \lambda_1 \lambda_2$ and $trace(M) = \lambda_1 + \lambda_2$. Using this scoring function, we can classify the window as follows:

- $|R|$ is small (eigenvalues) are small, the region is flat
- $R < 0$ the region contains edge ($\lambda_1 >> \lambda_2$ or vice versa)
- R is large, then the region contains an edge.

using a threshold, we can automate this process in our codes.

The Shi-Tomasi algorithm uses the same procedure however, instead of the Harris scoring function will use the following scoring function:

$$R = min(\lambda_1, \lambda_2) \tag{6}$$

which requires an estimation of the minimum eigenvalue.

Maximally Stable Extremal Regions (MSER) Overview

Maximally Stable Extremal Regions (MSER) is a robust feature detection algorithm commonly used in computer vision for object detection and region-based segmentation tasks and has also been used in our code due to its effectiveness in detecting and generating features to be tracked, enjoying robustness with respect to affine transformation.

MSER operates by detecting regions within an image that remain stable across a wide range of intensity thresholds. These regions, referred to as extremal regions, are areas where all pixels within the region are either brighter or darker than the surrounding pixels.

Formally, an extremal region R in an image I can be defined as a connected component of the set of pixels P in I, where the intensity of all pixels within R is either strictly greater than (bright extremal regions) or strictly less than (dark extremal regions) the intensity of the boundary pixels of R. The stability of a region is evaluated based on the relative change in the size of R as the intensity threshold varies.

Consider an image $I(x, y)$, where (x, y) represents pixel coordinates. For a given intensity threshold t, an extremal region R_t is defined as:

$$R_t = \{(x, y) \mid I(x, y) \leq t\} \quad \text{(for dark regions)}$$
$$R_t = \{(x, y) \mid I(x, y) \geq t\} \quad \text{(for bright regions)}$$

The region R_t is considered maximally stable if the relative rate of change of its area $|R_t|$ with respect to the change in threshold t is minimal. This stability criterion can be mathematically expressed as:

$$\frac{|R_{t+\Delta t}| - |R_t|}{|R_t|} \quad \text{is minimal for a stable region, where } \Delta t \text{ is a small intensity step.}$$

MSER is particularly effective in detecting blob-like structures and text regions in images, as these areas typically remain stable over a range of thresholds. This stability makes MSER robust to lighting changes and affine transformations, which are common challenges in real-world applications. In practical terms, MSER identifies regions that can be used as features for tasks such as object recognition, image matching, and text detection.

The algorithm efficiently extracts regions by incrementally adjusting the intensity threshold and evaluating the stability of each candidate region. Only those regions that exhibit maximal stability are retained, reducing the number of features and focusing on the most reliable ones.

It should be noted that MSER is also not supported by Cuda OpenCV, so we are dependent on CPU-based accelerations mentioned in the previous chapter to accelerate this part of the code.

Other Feature Detection Approaches

It is worth noting that the above feature detection algorithms are not accessible by Python OpenCV with Cuda. Therefore, alternatively, one can use other Cuda-enabled feature detection algorithms. As of the latest updates, OpenCV with CUDA support provides GPU-accelerated implementations of several key feature detection and description algorithms that include:

- ORB (Oriented FAST and Rotated BRIEF): A fast and efficient feature detector and descriptor extractor that combines the FAST key point detector and BRIEF descriptor.
- FAST (Features from Accelerated Segment Test): A fast feature detector that is particularly good for detecting corners in images.
- BRISK (Binary Robust Invariant Scalable Keypoints): A scale and rotation-invariant feature detector and descriptor extractor.
- SURF (Speeded-Up Robust Features): A robust feature detector and descriptor extractor, particularly good for handling scale and rotation variations (note that SURF is patented and may require a license for commercial use).

These Cuda-accelerated implementations help to significantly speed up the feature detection and matching processes, especially when working with large images or video streams.

Global Motion Estimation

Global motion estimation (GME) is one of the primary steps for distinguishing between ego-motion and the movement of objects within the scene. In fact, global motion estimation determines the background motion cluster. Various methods have been proposed to tackle this challenge. The proposed approaches mostly work by tracking salient features or regions between consecutive frames. Here, in our work, the approach discussed in [23]. One can also refer to [24, 25] as related works in this regard.

Motion estimation based on the affine transformation between two frames: In this method, firstly, the camera motion will be modeled by a 6-parameter affine transformation projected on the image coordinate frame:

$$\begin{cases} dx = a_0 + a_1 x + a_2 y \\ dy = a_3 + a_4 x + a_5 y \end{cases} \tag{7}$$

This transformation can be written in the following matrix form:

$$\begin{bmatrix} dx \\ dy \end{bmatrix} = \begin{bmatrix} 1 & x & y & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x & y \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = A(x,y)\mathbf{a} \tag{8}$$

It should be noted that affine transformations are very general, including translation, scaling, homothety, similarity, reflection, rotation, shear mapping, and compositions in any combination and sequence, which can, in fact, describe most of the possible events and transitions in between video frames.

Two approaches can be adopted for identifying the above model. In the first approach also called the direct approach, the model will be identified such that the transformation minimizes the following cost function:

$$E = \sum_{x,y} [I_1(x + dx, y + dy) - I_2(x, y)]^2 = \sum_{x,y} (e(x, y))^2 \tag{9}$$

over all the pixels. Obviously, this will soon cause computational complexity, leading to slow-motion identification that limits the applicability of this approach for real-time applications. In another approach called the indirect approach, the error between the affine model motion of features will be minimized. For this purpose, we need to first calculate the features' motion in between consecutive frames, which can be done using sparse optical flow algorithms for tracking feature points or through block matching algorithms such as CAMShift. A multi-scale indirect algorithm has been proposed in [23] for global motion estimation and has been followed in our work as well.

It is worth noting that the camera motion is, in fact, the result of a continuous motion of the camera base, which in our case is due to drone motion and the camera motion with respect to the drone body. Considering this, we can impose further continuity constraints by applying an auto-regressive (AR) filter or using Kalman filtering based on drone and camera motion kinematics to the motion parameters.

It is worth mentioning that in cases where the data of Inertial measurement units (IMUs) are available, that information can also be properly transformed to the camera coordinate frame using the available kinematic relations discussed before to increase the accuracy. Unfortunately, in this project, we could not easily use the IMU information. Therefore, we ignored this method.

The calculated motion can either be used directly as a feature in the remaining steps of moving object detection or compensated for the camera motion before calculating the optical flows by affine wrapping the consecutive frames. The latter can be done by using interpolated frames between two consecutive frames, which increases the dependency of the optical flows on object motions rather than the camera motion. In contrast, in the first approach, one can first convert the calculated affine transformation into optical flow vectors assigned to pixels and then use this optical flow map as a cluster center for the background cluster.

It is worth mentioning that the affine transformation model cannot be used for describing particular $3D$ and $2D$ motions, other possible motion models that can be used here include:

- Quadratic Model, which is a $2D$ model;
- Planar Projective Transform or Homograph model, which is again a $2D$ model;
- Homograph plus epipole model, which is a $3D$ model;
- plane and parallax model that is a $3D$ model.

Optical Flow Calculation

Images are $2D$ signals and have limitations in capturing the events in a $3D$ world, to better understand these limitations in this section, first, we will go through the kinematic relations between images and the $3D$ world coordinate frame, from which we can better define the optical flows and understand their limitations in finding the structure of the motion.

Consider a $3D$ coordinate frame $r_W = (X_W, Y_W, Z_W)$ describing the position of points in the environment. The world coordinate frame will be considered to be inertial and fixed for the rest of the discussions. We assign a coordinate frame to the mounting base of the camera, which would be the same as the drone body frame, where a point will be described in this frame using $r_B = (X_B, Y_B, Z_B)$. The camera is also associated with a coordinate frame called a camera coordinate frame, where the point in this frame will be denoted by $r_C = (X_C, Y_C, Z_C)$. The camera projects the points into a $2D$ plane, which is associated with an image coordinate frame where the points in this frame will be described by (x, y).

The world, body, and camera-coordinated frames are related to each other through homogeneous transformations. In particular, we have:

$$\begin{bmatrix} r_B \\ 1 \end{bmatrix} = H^B_{BW} \begin{bmatrix} r_W \\ 1 \end{bmatrix}, \begin{bmatrix} r_C \\ 1 \end{bmatrix} = H^C_{CB} \begin{bmatrix} r_B \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} r_C \\ 1 \end{bmatrix} = H^C_{CW} \begin{bmatrix} r_W \\ 1 \end{bmatrix} = H^C_{CB} H^B_{BW} \begin{bmatrix} r_W \\ 1 \end{bmatrix} \tag{10}$$

where we have:

$$H^X_{XY} = \begin{bmatrix} Rot(\phi, \theta, \psi) & p^X_{O_{XY}} \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{11}$$

where $Rot(\phi, \theta, \psi)$ is the general $3D$ rotation matrix describing the orientation of the current coordinate frame with respect to the destination frame, and $p^X_{O_{XY}}$ is the vector described in the destination frame describing the position of the origin of the current frame in destination frame.

The transformation from the camera frame to the image frame then will be described through the camera calibration matrix as well as a projection into u and v, as follows:

$$\begin{cases} u = \alpha \frac{X_C}{Z_C} - \alpha \cot(\vartheta) \frac{Y_C}{Z_C} + c_x \\ v = \frac{\beta}{\sin(\vartheta)} \frac{Y_C}{Z_C} + c_y \end{cases} \tag{12}$$

To analyze the motion projection on the image coordinate, we have that:

$$\begin{cases} \dot{u} = -\alpha \frac{\dot{X}_C Z_C - \dot{Z}_C X_C}{Z_C^2} + \alpha \cot(\vartheta) \frac{\dot{Y}_C Z_C - \dot{Z}_C Y_C}{Z_C^2} = -\alpha \frac{\dot{X}_C}{Z_C} + \alpha \frac{\dot{Z}_C}{Z_C} \frac{X_C}{Z_C} + \alpha \cot(\vartheta) \frac{\dot{Y}_C}{Z_C} - \alpha \cot(\vartheta) \frac{\dot{Z}_C}{Z_C} \frac{Y_C}{Z_C} \\ \dot{v} = -\frac{\beta}{\sin(\vartheta)} \frac{\dot{Y}_C Z_C - \dot{Z}_C Y_C}{Z_C^2} = -\frac{\beta}{\sin(\vartheta)} \frac{\dot{Y}_C}{Z_C} + \frac{\beta}{\sin(\vartheta)} \frac{\dot{Z}_C}{Z_C} \frac{Y_C}{Z_C} \end{cases} \tag{13}$$

which can be written as:

$$\begin{cases} \dot{u} = -\alpha \frac{\dot{X}_C}{Z_C} + \alpha \cot(\vartheta) \frac{\dot{Y}_C}{Z_C} - \dot{Z}_C \frac{u - c_x}{Z_C} \\ \dot{v} = -\frac{\beta}{\sin(\vartheta)} \frac{\dot{Y}_C}{Z_C} - \dot{Z}_C \frac{v - c_y}{Z_C} \end{cases} \tag{14}$$

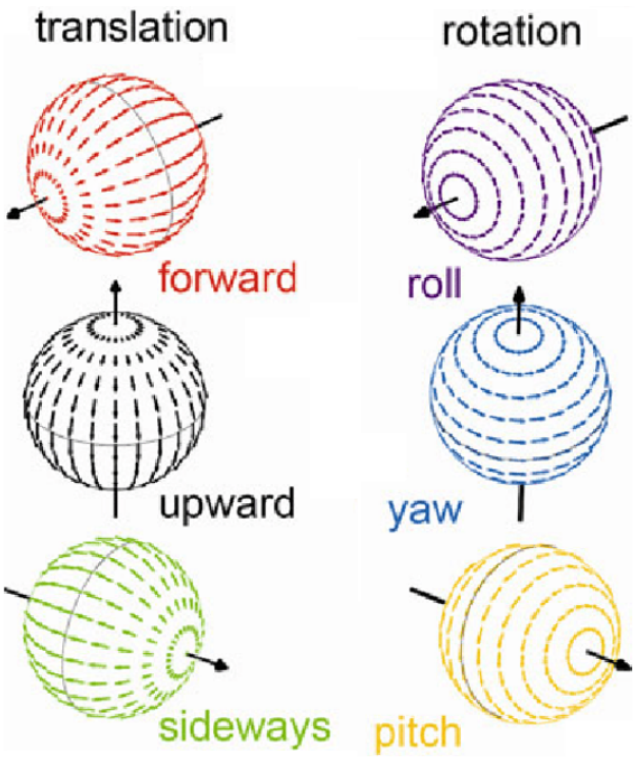


Figure 8: Optical flow patterns resulting from different motions perceived by the camera

The kienamtic of motion of $(X_C, Y_C, Z_C)^T$ with respect to world coordinate frame can be written as $(\dot{X}_C, \dot{Y}_C, \dot{Z}_C)^T = P(t) + \omega \times (X_C, Y_C, Z_C)^T$, which can be replaced in the above equations. Discretizing the vector $(\dot{u}, \dot{v})$ given the frame rate $f_s = 1/T_s$, we have the projected displacement as $(\dot{u}T_s, \dot{v}T_s)$, which describes the projected motion field on the image coordinate.

It is worth investigating the projection of different types of motion on optical flow maps. The following figure 8 summarizes the patterns resulting from different camera motions.

Within the past decades, a wealth of different optical flow generation algorithms for motion analysis in video streams have been developed, including both analytical approaches based on solving optimization problems and using machine learning architectures. We refer the reader to [26]. There are also hardware-supported optical flow algorithms, such as Nvidia optical implementation on Turing's GPU, which is accessible through OpenCV.

In particular, in this work, we have tested the following optical flow algorithms:

- FarnebackOpticalFLow, implemented by GPU along with mean shift filtering to reduce the texture in the image before evaluating the optical flows.
- DISOpticalFlow, which is the fastest among all of them, and has been implemented with CPU.
- BroxOpticalFlow implemented with GPU, which is supposed to be among the most accurate analytic methods for optical flow generation.
- OpticalFlowDual_TVL1, which even its GPU implementation is very slow.
- DensePyrLKOpticalFlow, this is the sparse to dense version of the Lucas-Kanade optical flow algorithm, which has been implemented with GPU. It is very fast, yet the accuracy is not as good as others, which was also an expectation.
- DeepFlow is a deep learning-based algorithm implemented in opencv_contrib; it is implemented with the CPU, and although its accuracy is fairly good, due to CPU implementation, the execution rate is very slow.

- SimpleFlow, which, despite what it has been claimed, its CPU implementation is very slow, even slower than DeepFlow.
- RAFT deep learning optical flow algorithm has been published in 2020 among the most accurate and efficient deep learning algorithms, its pytorch implementation with GPU has been tested in this project, and however its accuracy, as claimed, is very good, its execution time is not comparable with those analytic methods like DISOpticalFlow, Farneback, or BroxOpticalFlow.

In general, optical flow algorithms can be classified into two categories: dense optical flows, which assign optical flows to each pixel of the video frames, and sparse optical flows, which assign optical flows to frame features such as edges and corners and track them in between frames. An example of dense optical flow algorithms is the Gunner-Farneback algorithm, and we have the Lucas-Kanade algorithm for calculating sparse optical flows.

Feature Selection

When using dense optical flow maps for motion detection and estimation with a moving camera, several features can be extracted to analyze the motion. The goal is to distinguish between the motion caused by the moving camera and the motion of objects in the scene. Below are some key features:

1. **Flow Magnitude** The magnitude of the optical flow vector at each pixel gives the speed of motion at that point. This can help in identifying regions with significant motion, which could indicate object motion or high camera movement areas.
2. **Flow Direction (Angle)** The direction of the flow vector at each pixel, typically measured in radians or degrees. Comparing the flow directions across the frame can help in distinguishing between the motion due to the camera and actual object motion. Uniform flow direction often indicates camera motion, while deviations may indicate object movement.
3. **Flow Divergence** The divergence of the optical flow field indicates areas where the flow vectors are converging or diverging. This feature can be useful for detecting approaching or receding objects relative to the camera. It can also indicate camera zoom effects.
4. **Flow Curl (Rotation)** The curl of the optical flow field measures the rotational component of the flow. This feature is particularly useful in detecting the rotational motion of objects or rotational effects caused by the camera's own motion.
5. **Local Flow Consistency** The consistency of flow vectors within local regions or superpixels. Uniformity in local regions might indicate background motion due to the camera, while inconsistencies could indicate object boundaries or motion.
6. **Temporal Flow Changes** Changes in optical flow over consecutive frames. This can be used to track objects over time and to detect acceleration or deceleration, helping to distinguish between moving objects and static backgrounds under camera motion.
7. **Histogram of Oriented Optical Flow (HOOF)** A histogram representing the distribution of flow vectors' directions. Useful for summarizing motion patterns and can be used in conjunction with classifiers to detect specific types of motion.
8. **Optical Flow Entropy** A measure of the uncertainty or variability in the optical flow distribution. High entropy regions may indicate complex motion, such as interactions between multiple moving objects or cluttered scenes.

9. **Optical Flow Coherence** The coherence of the flow field, indicating how well the flow vectors align with each other. Regions with coherent flow might correspond to rigid body motion, while incoherent regions might indicate deformable or articulated objects.

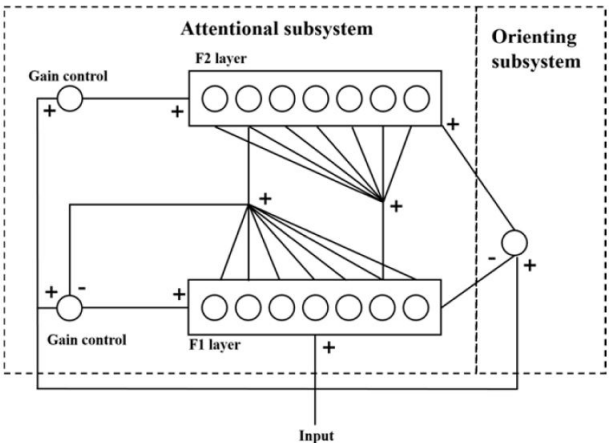
These features, along with the salient features of the frames and the camera ego-motion estimate, can be used for moving object detection and motion segmentation.

ART2 Neural network

Adaptive resonance theory is a type of neural network technique developed by Stephen Grossberg and Gail Carpenter based on how the brain autonomously learns to attend, categorize, recognize, and predict objects and events in a changing world [27]. ART uses unsupervised learning techniques to address the problem of pattern recognition and prediction. ART1 is used for clustering binary vectors, and ART2 recognizes either binary or analog inputs. Adaptive resonance theory networks give users the ability to control the degree of similarity of patterns by using the relative similarity of an input pattern to the weight vector on the constant cluster. The ART networks are designed to be stable and plastic. The plasticity here means that our network can learn new patterns at any stage of the learning process and has the flexibility to receive any new information, and with this new information, the system remains stable [28].

The basic architecture of an adaptive resonance network consists of three classes of neurons: an input processing field (F_1 layer) and the cluster units (F_2 layer) in the attentional subsystem, which consists of a comparison layer and a recognition layer. In the orienting subsystem, gains try to regulate and manage the ART network by controlling the degree of similarity of the pattern assigned to the same cluster (reset mechanism). Gain controls and the reset mechanism provide control functions needed for training and classification.

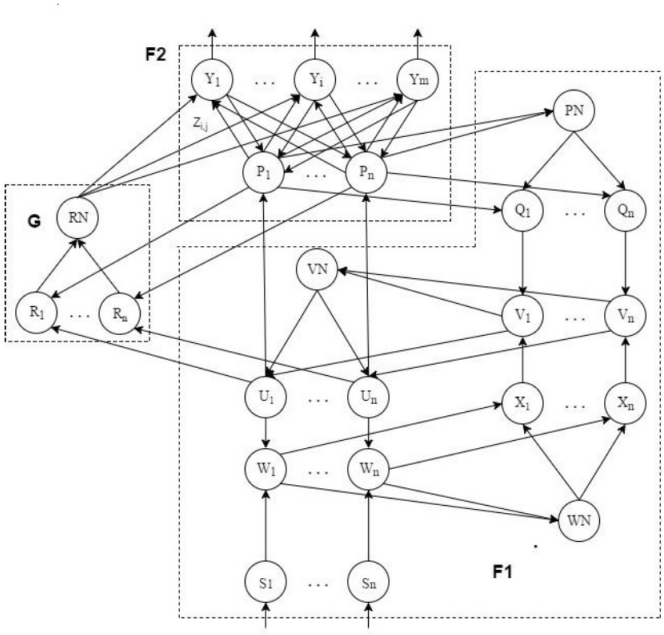
For controlling the similarity of patterns on the same cluster, two types of weights are considered. Bottom-up weights which connect F_1 layer to F_2 layer and top-down weights which is connected F_2 layer to F_1 layer. The F_2 layer is a competitive layer, and in this layer, the cluster units with the largest net inputs become the winners, and the rest of the activation neurons are set to zero. Based on the similarity between the input vector and the top-down weight vector, the cluster unit is allowed to learn the input pattern. The interface unit then merges information from the input units and cluster units. The reset unit decides, based on the input and interface unit, that if the cluster is not authorized to learn, then it is inhibited, and a new cluster unit is selected.



The structure of the adaptive resonance theory (ART) network

The ART2 network works with continuous values of inputs, which is the same as what ART1 does for binary inputs. The more complicated F_1 layer is needed in ART2 since its continuous input data

may be close together. It should be noticed that the path from F_1 to F_2 includes a Long Term Memory (LTM) trace or adaptive weights. Figure 2 demonstrates ART2 architecture, and it consists of three main fields of F_1, F_2, and G. The F_1 layer consists of ten types of units F_1 = S, W, X, V, U, Q, P, WN, VN, PN where the first seven elements are neurons layer for processing ART2 and the last three elements have the task of normalizing. The action of F_2 layer is the same as ART1 and consists of recognizing neurons Yj (j = 1…m, where m is the number of different classes of images), F_2= Y. In this stage, the units compete in a winner-take-all mode to learn each input pattern. The learning occurs only when the top-down weight vector for the winning unit is adequately similar to the input vector. Field G is a tuple consisting of control neurons R_i (where i = 1..n, and n – number of input parameters), G = R, RN, where RN normalizing element, and it should be noticed that the tests for reset in ART1 and ART2 are different [29].



ART2 Architecture [29]

The following stages provide a mathematical description of the implementation of the ART2 network in our work.

- **Stage 1**: The input of S-layer $s = (s_1, ..., s_i, ..., s_n)$ is used in the learning trail as a presentation of an input pattern. It should be considered that at the beginning of the learning trial, all activation is set to zero. Then, the signals of the S-layer are received at the W-layer and added to the output signals of the U-layer.

$$\begin{aligned} u_i &= 0, \\ w_i &= s_i + a u_i \qquad i = 1, ..., n \end{aligned} \tag{15}$$

- **Stage 2**: The output signals w_i of the W-layer carry on to the inputs of the X-layer and normalizing element.

$$\begin{aligned} \|w\| &= \sqrt{\sum_{i=1}^{n} w_i^2}, \\ x_i &= \frac{w_i}{e + \|w\|}, \qquad i = 1, ..., n, \end{aligned} \tag{16}$$

where the parameter e here is a small positive constant, which prevents the division by zero.

- **Stage 3**: Now the output signals of the V-layer are computed:

$$\nu_i = f\left(x_i\right) + bf\left(q_i\right) \qquad i = 1, ..., n \quad q_i = 0 \tag{17}$$

The activation function considering the noise signal is defined below, and θ is the noise threshold:

$$f(x) = \begin{cases} x & \text{if } x > \theta \\ 0 & \text{else} \end{cases} \tag{18}$$

- **Stage 4**: The output signal of the previous layer ν_i from the V-layer passes to the input of the U-layer and into the normalizing element VN, as follows

$$\|\nu\| = \sqrt{\sum_{i=1}^{n} \nu_i^2},$$
$$u_i = \frac{\nu_i}{e + \|\nu\|}, \qquad i = 1, ..., n, \tag{19}$$

- **Stage 5**: The signals u_i will determine the signals p_i like before we have normalizing element PN

$$\|p\| = \sqrt{\sum_{i=1}^{n} p_i^2},$$
$$p_i = \frac{p_i}{e + \|p\|}, \qquad i = 1, ..., n, \tag{20}$$

The update of F_1 unit has already occurred, and we are trying to compute signals for F_2.

- **Stage 6**: To compute signals to F_2 units, the output signals from the P-layer pass to the recognition layer Y, such as

$$y_j = \sum_{i=1}^{n} z_{ij} p_i, \quad j = 1, ..., m. \tag{21}$$

Notice that in this stage, we find F_2 unit Y_j with the largest signal.

$$i \max = \max\left(Y\right). \tag{22}$$

The output signal from the Y-layer according to Eq.(22) is calculated.

$$p_i = u_i + t_{Ji} d, \quad i = 1, ..., n, \tag{23}$$

where t weight is related to the P-layer and Y-layer.

- **Stage 7**: For the reset mechanism, we have to calculate the signals of the control layer as below

$$r_i = \frac{u_i + cp_i}{e + \|u\| + c\|p\|}. \tag{24}$$

The normalizing element RN is also calculated

$$\|r\| = \sqrt{\sum_{i=1}^{n} r_i^2}. \tag{25}$$

- **Stage 8**: Now we check for rest condition, which is if $\|r\| \geq \rho - e$. Then, the assumption of the winner neuron is correct.

$$
\begin{aligned}
w_i &= s_i + au_i, \\
x_i &= \frac{w_i}{e + \|w\|}, \\
q_i &= \frac{p_i}{e + \|p\|}, \\
\nu_i &= f\left(x_i\right) + bf\left(q_i\right). \qquad (26)
\end{aligned}
$$

If $\|r\| \leq \rho - e$, then current active neuron is inhibited where $y_J = -1$.

- **Stage 9**: After checking the rest condition, we are going to update the weights of the winning unit.

$$t_{Ji} = \alpha d u_i + \{1 + \alpha d\left(d - 1\right)\} t_{Ji}, \qquad (27)$$

$$b_{iJ} = \alpha d u_i + \{1 + \alpha d\left(d - 1\right)\} b_{iJ}. \qquad (28)$$

- **Stage 10**: The F_1 activations will be updated as below

$$
\begin{aligned}
u_i &= \frac{\nu_i}{e + \|\nu\|}, \\
w_i &= s_i + au_i, \\
p_i &= u_i + dt_{Ji}, \\
x_i &= \frac{w_i}{e + \|w\|}, \\
q_i &= \frac{p_i}{e + \|p\|}, \\
\nu_i &= f\left(x_i\right) + bf\left(q_i\right). \qquad (29)
\end{aligned}
$$

Testing the stop conditions for weight updates and a number of epochs are done in the final learning stage.

Chapter-V: Experimental Results and Future Directions

In this chapter, the performance of the proposed solutions will be evaluated through the evaluation of the implementation of the processing pipeline on the VisDrone dataset and KITTI datasets.

We have tested the results over both the VisDrone Multi-object tracking sequences and the KITTI 2015 flow datasets, comparing the detected clusters across the entire sequence with the ground truth. For this purpose, a bounding box has been generated based on each cluster center and the distribution of pixels belonging to each cluster. The results have then been compared with the ground truth bounding boxes to evaluate false positives, true negatives, true positives, and false negatives.

After generating the bounding boxes, we matched the masked block with the annotated frames to establish the relationship between the cluster labels and the object IDs. We then evaluated the ratio of the area of intersection between the bounding boxes (one based on the ground truth and one based on the proposed algorithm) to the area of the exclusive union of the bounding boxes. If this ratio exceeds a pre-selected threshold, the detection is considered as true; otherwise, it is rejected.

This way we are able to generate the confusion matrix to evaluate the performance over each of the sequences. In particular:

A True Positive occurs when the algorithm correctly detects an object that is present in the ground truth, meaning that the generated bounding box sufficiently overlaps with the ground truth bounding box. For example, the algorithm detects an object in a video frame, and the bounding box generated by the algorithm overlaps well with the ground truth bounding box, and the objects can be matched. This detection is a true positive.

A False Negative happens when the algorithm fails to detect an object that is present in the ground truth. Either no bounding box is generated, or the bounding box does not meet the overlap threshold with the ground truth. There is an object in the ground truth annotations, but the algorithm does not generate a bounding box around that object, or the generated box does not overlap sufficiently with the ground truth box. This miss is a false negative.

A False Positive occurs when the algorithm incorrectly identifies an object where none exists in the ground truth, generating a bounding box in a region that should not contain any object. For example, the algorithm generates a bounding box in a region where there is no actual object according to the ground truth, such as on an empty road or in a background area without any relevant object. This incorrect detection is a false positive.

Precision: Precision measures how many of the objects detected by the algorithm were actually correct. It is the ratio of true positives to the total number of objects detected (true positives + false positives).

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall: Recall measures how many of the actual objects present were correctly detected by the algorithm. It is the ratio of true positives to the total number of actual objects (true positives + false negatives).

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Finally, the F_1 score can be calculated as follows:

$$F_1 = 2\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Examples of detection results, the generated bounding box based on $5D$ feature clustering, and the ground truth bounding boxes have been depicted in the figure 9; we also highlighted examples of missed detection.

(a) Detection Results: Blue Boxes are the motion detection; Black boxes are the annotation boxes

(b) Detection Results: Blue Boxes are the motion detection; Black boxes are the annotation boxes. The circles highlight instances of missed detection

Figure 9: Examples of detection results

VisDrone Test:

To evaluate the effectiveness of the algorithm, we conducted a series of experiments on the ViS-Drone dataset, which consists of sequences captured by drones in various urban and rural environments. The dataset includes a wide range of challenges, such as varying object scales, motion blur, and occlusions.

The best results have been achieved by using the Brox Optical flow algorithm, and the magnitude and phase of the optical flows along with optical flow curl for each pixel coordinate as the features for clustering. The clustering algorithm has been configured with the following parameters:

- **Number of Neurons:** *[15 initially, dynamically will be adjusted up to maximum 25]*
- **Vigilance Parameter (ρ):** *[0.2]*
- **Learning Rate (β):** *[0.95]*

These parameters were chosen based on initial experiments with a subset of the dataset to balance sensitivity to new patterns and stability in learning, as well as based on the typical number of objects within each video stream.

The achieved performance is summarized in figure 10.

The algorithm provides a reasonable balance between precision and recall, with an F1-Score that indicates moderate performance across different sequences in the VisDrone dataset. However, the variability in metrics suggests that the algorithm's robustness should be improved, particularly in more challenging sequences where false positives are more common and recall is lower. Additionally,

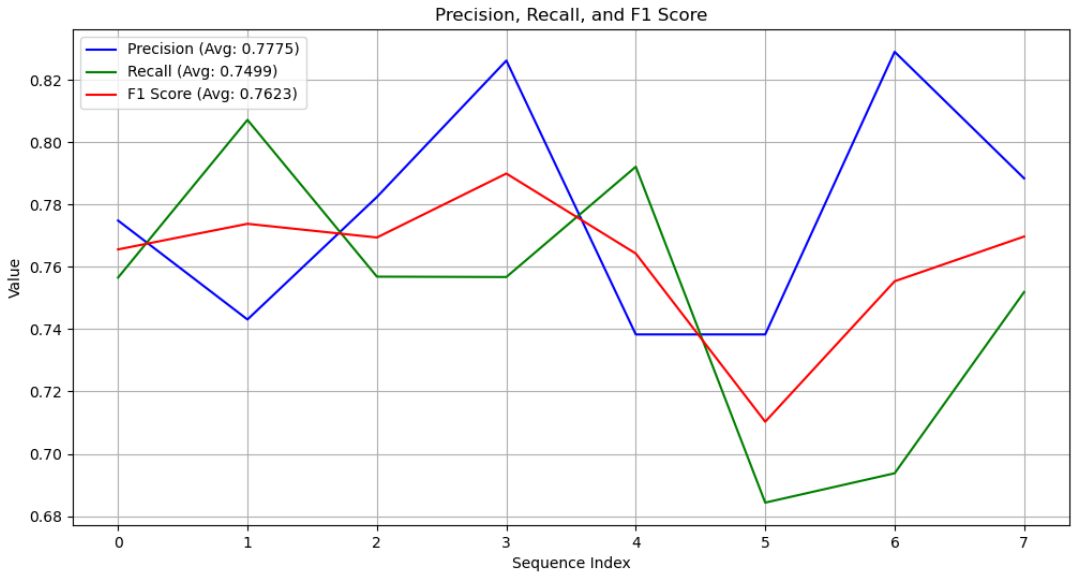


Figure 10: Performance metric of the proposed algorithms evaluate over VisDrone Multi-object tracking dataset

the processing time is acceptable for real-time applications, making this approach viable for practical use, albeit with the potential for further optimization.

The average processing time was approximately 60 ms per frame (frame rate of 16), making the algorithm suitable for almost real-time applications, particularly in scenarios where a small delay is acceptable. The processing time varied slightly across sequences due to differences in frame complexity and the number of detected objects.

The frame rate can increase by downsampling the frames; instead of applying the algorithm to each frame, we can apply it to every two out of five frames. Obviously, with this latter approach, the frame rate would be increased, yet, due to larger motion being experienced, the motion segmentation results will reduce.

In the sequel, the outputs for three sample sequences, i.e., sequences 2, 5, and 4, have been investigated in more detail.

In the first sequence, the drone is moving slowly and has a linear motion with a large portion of the frames, for which the affine camera motion model gives better performance. However, in this sequence, the object scales are small compared with the field of view with low separation. The last set of frames experienced a severe camera rotation, resulting in a performance drop that suggests the algorithm might still be sensitive to camera motions.

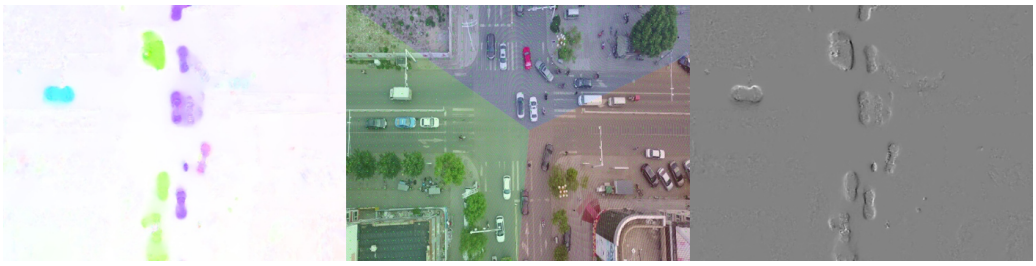
The following figures depict the original frames with an overlay of background optical flow vectors representing the estimated affine motion as an RGB overlay, along with the optical flow features,i.e. magnitude and angle, described as an RGB image in the left, and optical flow curls at each pixel described as a grayscale image in the right.

The performance in sequence 4 was moderate due to the small scale, high number, and low separation of the objects, which resulted in frequent conclusions; besides, the drone motion was more sensible in this case and also included a minor rotational motion.

The last sequence provided the lowest performance metrics due to harsher drone motions and the complexity of the scene.

(a) Image 1

(b) Image 2



(c) Image 3

(d) Image 4

Figure 11: Test Results for Video stream-I of VisDrone dataset of Multi-Object Tracking

(a) Image 1

(b) Image 2

(c) Image 3

(d) Image 4

Figure 12: Test Results for Video stream-II of VisDrone dataset of Multi-Object Tracking

(a) Image 1

(b) Image 2

(c) Image 3

(d) Image 4

Figure 13: Test Results for Video stream-III of VisDrone dataset of Multi-Object Tracking

The algorithm with RAFT optical flow demonstrated better performance in moving object detection on the VisDrone dataset, particularly in terms of precision; however, the processing frame rate is not acceptable, reaching 2 frames per second. The results indicate that the algorithm is well-suited for scenarios where false positives must be minimized, such as in surveillance or traffic monitoring. However, the slightly lower recall suggests that further refinement is needed to improve detection in challenging conditions, such as when the drone motion is complex.

KITTI Test

To evaluate the algorithm's effectiveness, we conducted a series of experiments on the KITTI dataset, which comprises sequences captured from a vehicle-mounted camera in urban, rural, and highway environments. The dataset poses several challenges, including varying object scales, motion blur, occlusions, and rapid camera motion due to the vehicle's movement. However, the objects usually have a larger scale with better separation than those of VisDrone.

The clustering algorithm was configured with the following parameters:

- **Number of Neurons:** *[5 initially, dynamically adjusted up to a maximum of 10]*
- **Vigilance Parameter (ρ):** *[0.5]*
- **Learning Rate (β):** *[0.95]*

These parameters were chosen based on initial experiments with a subset of the dataset to balance sensitivity to new patterns and stability in learning, taking into account the typical number of objects and the complexity of scenes within each video stream.

The achieved performance over the 28 video sequences of the KITTI dataset has been depicted in figure 14

The algorithm provides a good balance between precision and recall, with an F1-Score indicating acceptable performance across different sequences in the KITTI dataset. The achieved performance is comparable with those reported in [30], although there are still intermediate parameters that can be fine-tuned to improve the results and robustness.

The average processing time was approximately 63 ms per frame, which is acceptable. However, making the algorithm suitable for real-time applications with an an acceptable small delay.

The algorithm exhibited better performance in moving object detection on the KITTI dataset. The results suggest that the algorithm is more suitable for scenarios where the object has larger scales, and the camera motion is closer to linear and affine motion.

In the sequel, the outputs for three sample sequences have been investigated in more detail. These are associated with sequences 13, 3, and 6.

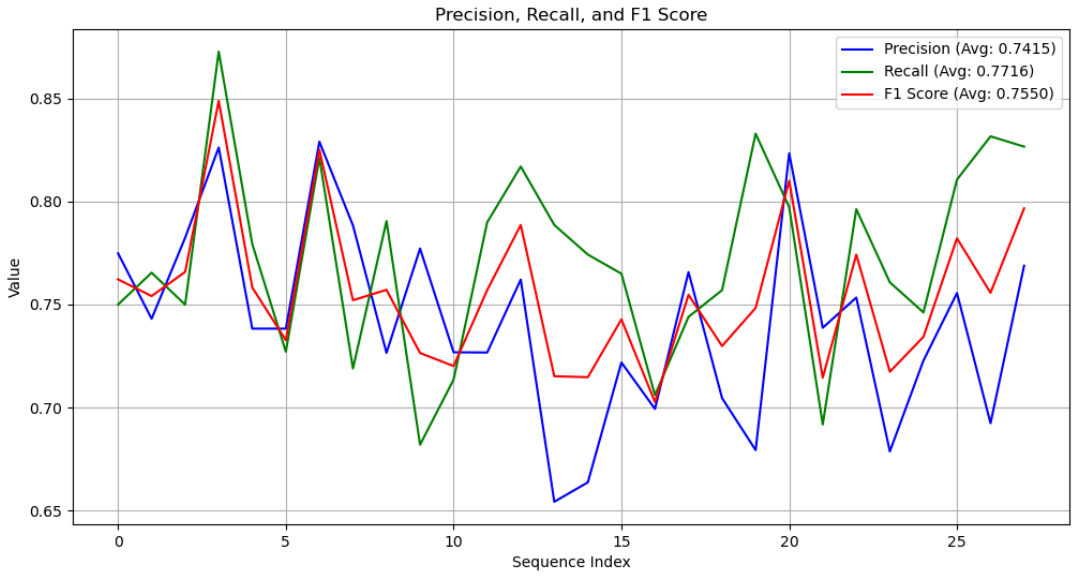
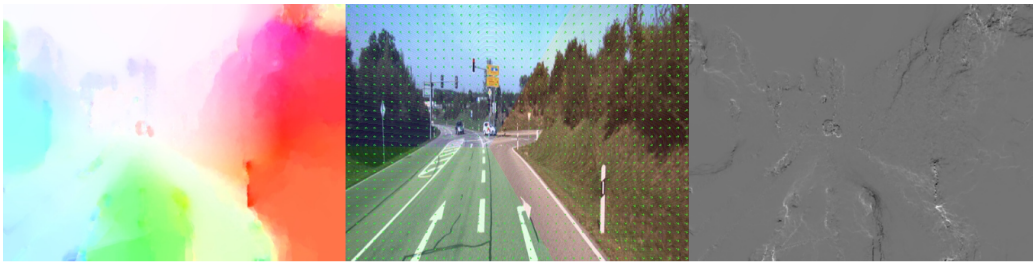


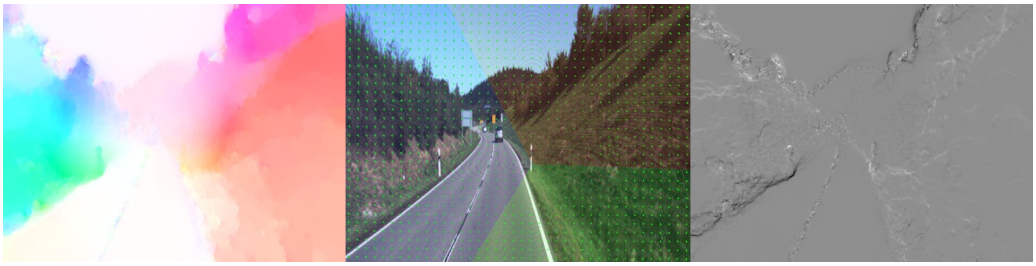
Figure 14: Performance metric of the proposed algorithms evaluate over KITTI



(a) Image 1

(b) Image 2

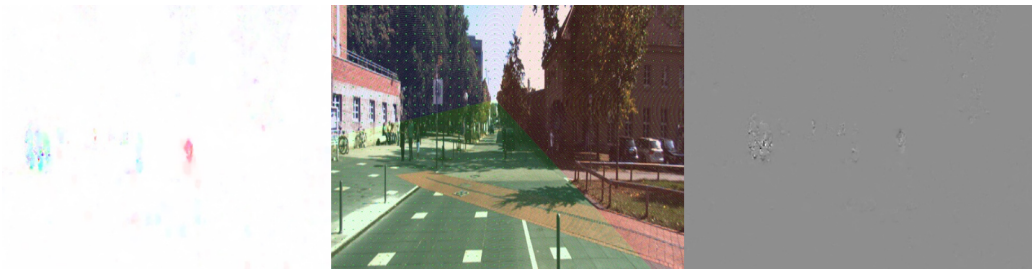
(c) Image 3



(d) Image 4

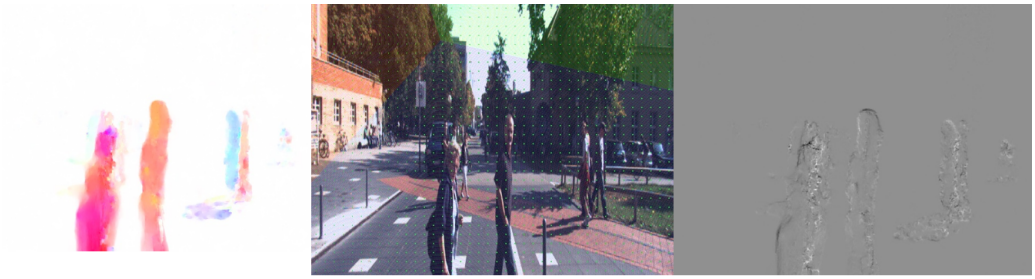
Figure 15: Test Results for Video stream-I of KITTI flow dataset

(a) Image 1



(b) Image 2

(c) Image 3



(d) Image 4

Figure 16: Test Results for Video stream-II of KITTI flow dataset

(a) Image 1

(b) Image 2

(c) Image 3

(d) Image 4

Figure 17: Test Results for Video stream-III of KITTI flow dataset

It is worth noting that in sequence 3 the camera was fixed, and therefore, the highest performance has been achieved. In contrast, the third scenario experienced camera motion within a similar scene, which slightly sabotaged the performance, yet, as the motion is close to affine transformation being used for global motion estimation, the variations remained acceptable.

Time Analysis of the Pipeline Components

Step	Time Taken (ms)	Description
Frame Preprocessing	14	Includes denoising, frame pyramid generation, and feature extraction.
Global Motion Estimation	11	Estimating the camera's motion relative to the background to differentiate from object motion.
Optical Flow Calculation	19	Computing optical flow to analyze pixel movement between frames.
Flow Clustering	16	Clustering optical flow vectors to identify and track moving objects.
Total	60	The average overall time taken on the VisDrone dataset.

Table 1: Time Analysis for VisDrone Dataset

Step	Time Taken (ms)	Description
Frame Preprocessing	13	Includes denoising, frame pyramid generation, and feature extraction.
Global Motion Estimation	16	Estimating the camera's motion relative to the background to differentiate from object motion.
Optical Flow Calculation	19	Computing optical flow to analyze pixel movement between frames,
Flow Clustering	15	Clustering optical flow vectors to identify and track moving objects,
Total	63	The average overall time taken on the KITTI dataset.

Table 2: Time Analysis for KITTI Dataset

To improve the performance of the proposed pipeline for moving object detection using a moving camera, several optimizations may be applied at each stage and to the overall system.

Frame Preprocessing: This step can be accelerated by optimizing the denoising and feature extraction processes. Implementing more efficient algorithms or utilizing GPU-based acceleration for tasks like Gaussian filtering and feature detection can significantly reduce processing time. Additionally, adaptive frame pyramid generation, which adjusts the resolution based on the motion complexity in the scene, can help balance between computational load and detection accuracy.

Global Motion Estimation: Performance can be enhanced by simplifying the motion models used or applying fast, approximate algorithms when exact precision is less critical. Integrating Inertial Measurement Unit (IMU) data, if available, can reduce computational overhead by providing initial estimates for motion that simplify the subsequent calculations. Employing hardware acceleration or optimizing code paths for specific hardware can also yield faster performance.

Optical Flow Calculation: Given its computational intensity, this stage benefits greatly from optimization. Using more efficient algorithms like DISOpticalFlow, which balances speed and accuracy helped in this work. Implementing optical flow calculations on the GPU or using parallel processing techniques can dramatically reduce the time required for this step. Moreover, applying these algorithms only to regions of interest identified in earlier steps rather than the entire frame can reduce unnecessary computations.

Flow Clustering with ART2: To speed up clustering, optimizing the ART2 algorithm by adjusting its parameters for faster convergence without sacrificing accuracy is key. Techniques like reducing the number of neurons in scenarios with simpler motion or implementing early stopping criteria for cluster updates can save processing time. Utilizing GPU-based implementations of neural networks can further improve the speed of this stage.

Overall Pipeline Optimizations: Across all stages, optimizing memory management by pre-allocating memory and minimizing data transfers between CPU and GPU can significantly reduce latency. Employing asynchronous processing allows overlapping computation and data transfer, maximizing resource utilization. Downsampling frames or processing a subset of frames rather than every single frame can also enhance overall speed, especially in real-time applications where slight delays are tolerable.

Conclusion and Future Directions

In this project, an algorithm for moving object detection with a moving camera has been implemented and tested.

Moving object detection and tracking with a moving camera is challenging due to the combined effects of the camera's motion relative to its mounting base and the movement of the platform within the environment. The limited field of view of the camera results in frequent object discontinuity and severe background variations, which conventional background subtraction approaches for fixed cameras cannot handle. To address these challenges, dense optical flow clustering is used to detect moving objects with a moving camera. The clusters correspond one-to-one with moving objects and background motion due to camera ego-motion; however, objects frequently enter or leave the scene, necessitating frequent redefinition and recalculation of clusters. Additionally, since the number of objects in the scene is unknown and can vary over time, the clustering algorithm must adapt quickly to changing scenarios. Therefore, we adapted the Adaptive Resonance Theory-2 (ART2) network to eliminate the need for pre-tuning the number of clusters as a hyper-parameter, automating the process similarly to human perception and enabling rapid redefinition and recalculation of varying cluster numbers. The performance of the proposed approach was evaluated in terms of execution time and accuracy using the VisDrone and KITTI datasets, and the results are discussed.

The achieved performance was more acceptable on the KITTI test than VisDrone, suggesting better applicability for collision avoidance applications that track while patrolling using drones. The algorithm is still sensitive to nonlinear camera motions and requires further adjustment for global motion estimation, potentially using $3D$ motion models instead of 2D models. Besides, there are hyper-parameters in the algorithm that should be adjusted regularly based on the current scene window, suggesting a recursive architecture for auto-tuning the code hyper-parameters being used. Moreover, the currently implemented algorithm does not benefit modestly from the history of the detected objects due to hard learning being used in the ART 2 network; this is while, after detecting an object and generating a template, this template can be used in non-parametric algorithms such as CAMShift to be tracked within the next frames, that automatically might improve the performance.

In this project several video coding technologies were applied to efficiently detect and track moving objects in video streams captured by moving cameras. Key technologies include CUDA for GPU acceleration, which is extensively used for computationally demanding tasks such as dense optical flow calculations. OpenCV, was leveraged with Intel-specific accelerations and CUDA- GPU support to implement various image processing algorithms. Several optical flow algorithms are employed, including Farneback Optical Flow, Brox Optical Flow, and RAFT. Decoding time has been considered in the evaluation, with the average processing time being approximately 60 ms per frame for VisDrone dataset and 63 ms for Kitti, making the algorithm suitable for almost real-time applications. The platform is designed to be flexible, supporting multiple video sources and coding formats, which allows it to adapt to various types of video streams and coding methods. It enhances its applicability in diverse scenarios, including recording and analyzing videos in real time with a mounted camera on a drone.

Bibliography

[1] A. Sobral and A. Vacavant, "A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos," *Computer Vision and Image Understanding*, vol. 122, pp. 4 – 21, 2014.

[2] L. Marcomini and A. Cunha, "A comparison between background modelling methods for vehicle segmentation in highway traffic videos," *arXiv preprint arXiv:1810.02835*, 2018.

[3] "How to use background subtraction methods." `https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html`.

[4] Z. Zivkovic and F. Van Der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern recognition letters*, vol. 27, no. 7, pp. 773–780, 2006.

[5] M.-N. Chapel and T. Bouwmans, "Moving objects detection with a moving camera: A comprehensive review," *arXiv preprint arXiv:2001.05238*, 2020.

[6] M. Yazdi and T. Bouwmans, "New trends on moving object detection in video images captured by a moving camera: A survey," *Computer science review*, vol. 28, pp. 157–177, 2018.

[7] W. Zhang, X. Sun, and Q. Yu, "Moving object detection under a moving camera via background orientation reconstruction," *Sensors*, vol. 20, no. 11, p. 3103, 2020.

[8] J. Huang, W. Zou, J. Zhu, and Z. Zhu, "Optical flow based real-time moving object detection in unconstrained scenes," *arXiv preprint arXiv:1807.04890*, 2018.

[9] R. Roberts and F. Dellaert, "Optical flow templates for superpixel labeling in autonomous robot navigation," in *5th Workshop on Planning Perception and Navigation for Intelligent Vehicles (PPNIV13)*, Citeseer, 2013.

[10] Q. Dong, C. Cao, and Y. Fu, "Rethinking optical flow from geometric matching consistent perspective," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1337–1347, 2023.

[11] Y. Zhang, G. Li, X. Xie, and Z. Wang, "A new algorithm for fast and accurate moving object detection based on motion segmentation by clustering," in *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*, pp. 444–447, IEEE, 2017.

[12] J. Ai, S. Wei, L. Zhang, and S. Sun, "A magnitude-based art2 classifier: structure and algorithms," in *2006 6th World Congress on Intelligent Control and Automation*, vol. 2, pp. 9799–9803, IEEE, 2006.

[13] J. Luo and D. Chen, "An enhanced art2 neural network for clustering analysis," in *First International Workshop on Knowledge Discovery and Data Mining (WKDD 2008)*, pp. 81–85, IEEE, 2008.

[14] J. He, A.-H. Tan, and C.-L. Tan, "Modified art 2a growing network capable of generating a fixed number of nodes," *IEEE Transactions on Neural Networks*, vol. 15, no. 3, pp. 728–737, 2004.

[15] J. Ai, B. Funt, and L. Shi, "A new type of art2 architecture and application to color image segmentation," in *Artificial Neural Networks-ICANN 2008: 18th International Conference, Prague, Czech Republic, September 3-6, 2008, Proceedings, Part I 18*, pp. 89–98, Springer, 2008.

[16] P. Zhu, L. Wen, D. Du, X. Bian, H. Fan, Q. Hu, and H. Ling, "Detection and tracking meet drones challenge," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7380–7399, 2021.

[17] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[18] "Image pyramids." https://docs.opencv.org/2.4/doc/tutorials/imgproc/pyramids/pyramids.html.

[19] P. Zhang and L. Xu, "Unsupervised segmentation of greenhouse plant images based on statistical method," *Scientific reports*, vol. 8, no. 1, pp. 1–13, 2018.

[20] R. Collins, "Lecture notes on harris corner detector," lecture notes, Pennsylvania State University, University Park, PA, USA, 2007.

[21] "Harris corner detector." https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html.

[22] "Shi-tomasi corner detector and good features to track." https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html.

[23] Samaretas, "Global motion estimation." https://github.com/Samaretas/global-motion-estimation, 2024. Accessed: 2024-08-08.

[24] Y. Wang, "Global motion estimation, video stabilization, panorama generation, motion deblurring."

[25] M. K. Olivier Adda, Nicolas Cottineau, "A tool for global motion estimation and compensation for video processing." https://users.encs.concordia.ca/~amer/teach/elec490/f02_rapport_final.pdf.

[26] M. Zhai, X. Xiang, N. Lv, and X. Kong, "Optical flow and scene flow estimation: A survey," *Pattern Recognition*, vol. 114, p. 107861, 2021.

[27] "Adaptive resonance theory." http://www.scholarpedia.org/article/Adaptive_resonance_theory.

[28] L. Fausett, *Fundamentals of neural networks: architectures, algorithms, and applications*. Prentice-Hall, Inc., 1994.

[29] D. Bukhanov and V. Polyakov, "An approach to improve the architecture of art-2 artificial neural network based on multi-level memory,"

[30] D. Zhou, V. Frémont, B. Quost, Y. Dai, and H. Li, "Moving object detection and segmentation in urban environments from a moving platform," *Image and Vision Computing*, vol. 68, pp. 76–87, 2017.