AI 503 - Introduction to Optimization
Project Report

for

Foreground - Background Separation

Github Repository:
https://github.com/kavyagupta3011/Foreground-Background-Separation

Prepared by:

1. Kavya Gupta (IMT2023016)

2. Nainika Agrawal (IMT2023034)

3. S. Divya (IMT2023059)

November 21, 2025

Contents

1 Abstract

This report presents an optimization-based approach to foreground-background separation using the CDnet 2014 dataset. We formulate the problem at two levels of complexity. The 'easy' problem (Mean/Median filtering) relies on temporal pixel statistics to build adaptive background models, suitable for static or mildly dynamic scenes. The 'hard' problem uses Robust Principal Component Analysis (RPCA), which adds complexity to more faithfully capture the original problem. RPCA is formulated as a constrained optimization problem that decomposes the video matrix into low-rank (background) and sparse (foreground) components.

2 Introduction

2.1 The Real World Problem and Motivation

In many real-world applications like video surveillance, robotics, and traffic monitoring, cameras capture both static backgrounds and moving objects. The primary purpose of this project is to develop and analyze methods to reliably separate these moving foreground elements from the static background.

Manually analyzing thousands of video frames is impractical and time-consuming. An efficient, automated solution has significant impacts, enabling:

- Accurate object detection and tracking by focusing only on separated foreground objects.
- Efficient background reconstruction, which reduces storage and computation by storing static scenes in a compact form.
- Faster analysis for quick, real-time video processing.
- Support for real-world applications in surveillance, autonomous driving, medical imaging, and traffic monitoring.

2.2 Why optimization?

A single video contains millions of pixel values that must be processed. Naive methods like subtraction or thresholding often fail in real-world scenarios due to challenges like fast movements and noise.

Optimization-based methods provide a structured way to:

- Extract clean foreground objects
- Preserve the background scene's structure
- Handle noise and variations efficiently
- Reliably separate background from moving objects

3 Dataset

We are using the **Change Detection Dataset (CDnet 2014)**, available on Kaggle. This dataset is particularly suited for testing our optimization methods as it covers both controlled and highly challenging scenarios.

- **Source:** `https://www.kaggle.com/datasets/maamri95/cdnet2014`
- **Purpose:** Designed for benchmarking algorithms in foreground–background separation and motion detection.
- **Content:** Video sequences from diverse and dynamic environments such as crowded areas, highways, and natural scenes.
- **Challenges Captured:** Includes moving people and vehicles, shadows, camera jitters, and other real-world variations.
- **Size & Scale:** Consists of 11 video categories, each containing 4 to 6 video sequences with 1,500+ frames each.
- **Resolution:** Frames vary between 320×240 and 720×486 pixels.
- **Key Variables:**
  - Input frames (video sequences)
  - Ground truth masks (per-pixel labels)

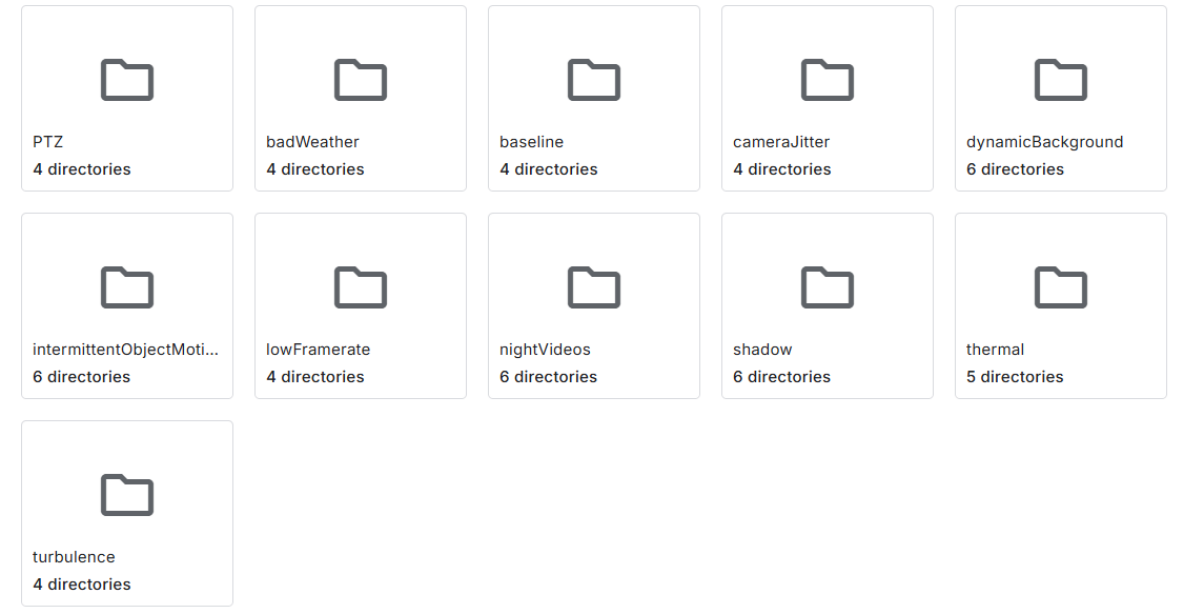
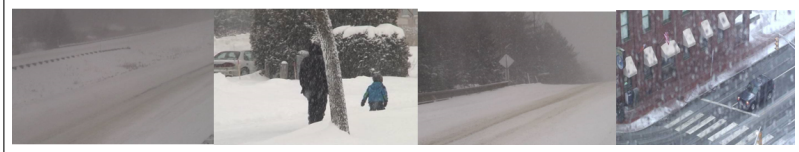


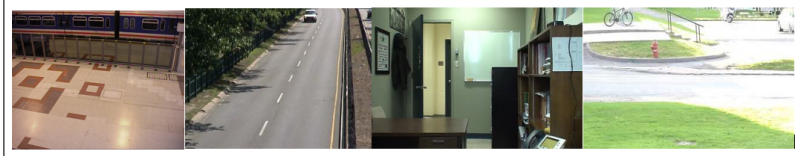
Figure 3.1: Dataset Folder Structure


PTZ (4 directories)
continuousPan
2 directories, 3 files
intermittentPan
2 directories, 3 files
twoPositionPTZCam
2 directories, 3 files
zoomInZoomOut
2 directories, 3 files


badWeather (4 directories)
blizzard
2 directories, 3 files
skating
2 directories, 3 files
snowFall
2 directories, 3 files
wetSnow
2 directories, 3 files

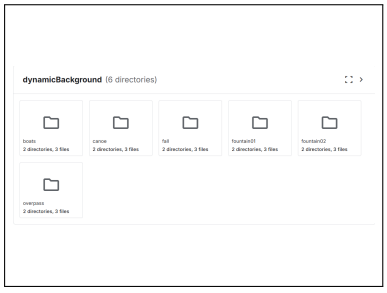



baseline (4 directories)
PETS2006
2 directories, 3 files
highway
2 directories, 3 files
office
2 directories, 3 files
pedestrians
2 directories, 3 files

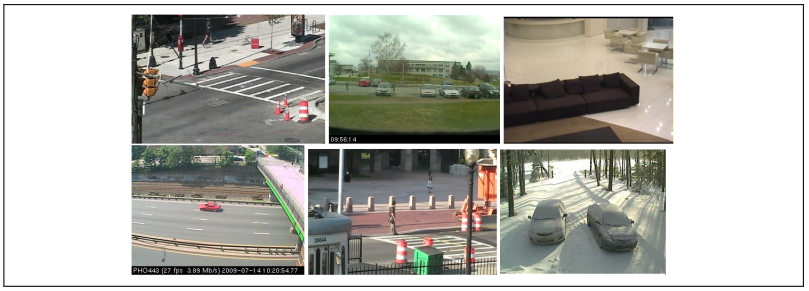


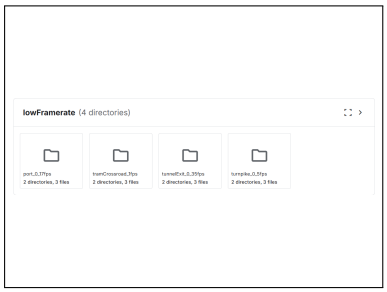

cameraJitter (4 directories)
badminton
2 directories, 3 files
boulevard
2 directories, 3 files
sidewalk
2 directories, 3 files
traffic
2 directories, 3 files

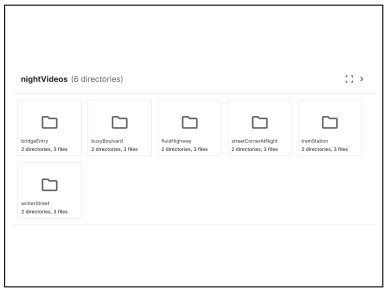

PHO West (30 fps, 8.17 Mb/s) 2008-08-01 11:49:43.33
PHO East (30 fps, 8.35 Mb/s) 2007-08-08 16:53:28.11

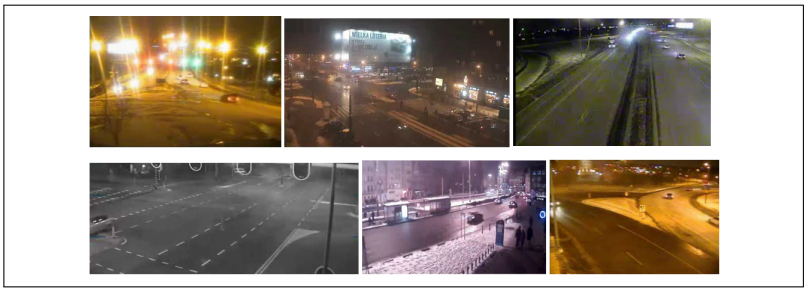

dynamicBackground (6 directories)
boats
2 directories, 3 files
canoe
2 directories, 3 files
fall
2 directories, 3 files
fountain01
2 directories, 3 files
fountain02
2 directories, 3 files
overpass
2 directories, 3 files

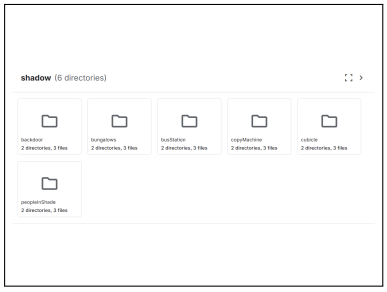

intermittentObjectMotion (6 directories)
abandonedBox
2 directories, 3 files
parking
2 directories, 3 files
sofa
2 directories, 3 files
streetLight
2 directories, 3 files
tramstop
2 directories, 3 files
winterDriveway
2 directories, 3 files

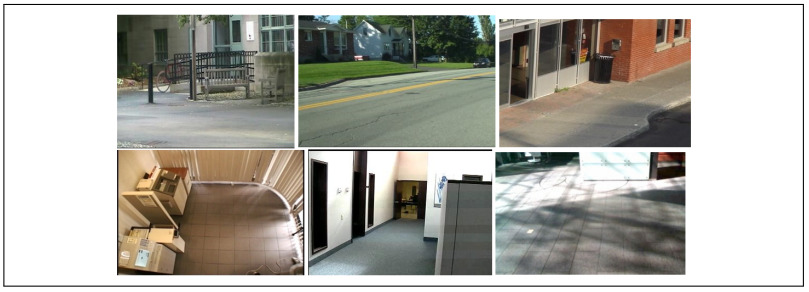



lowFramerate (4 directories)
port_0_17fps
2 directories, 3 files
tramCrossroad_1fps
2 directories, 3 files
tunnelExit_0_35fps
2 directories, 3 files
turnpike_0_5fps
2 directories, 3 files


nightVideos (6 directories)
bridgeEntry
2 directories, 3 files
busyBoulevard
2 directories, 3 files
fluidHighway
2 directories, 3 files
streetCornerAtNight
2 directories, 3 files
tramStation
2 directories, 3 files
winterStreet
2 directories, 3 files




shadow (6 directories)
backdoor
2 directories, 3 files
bungalows
2 directories, 3 files
busStation
2 directories, 3 files
coppMachine
2 directories, 3 files
cubicle
2 directories, 3 files
peopleInShade
2 directories, 3 files




thermal (5 directories)
corridor
2 directories, 3 files
diningRoom
2 directories, 3 files
lakeSide
2 directories, 3 files
library
2 directories, 3 files
park
2 directories, 3 files

Each video directory contains the following components:

- input/: A sub-directory containing the input video frames as individual .jpg files.
- groundtruth/: A sub-directory containing the ground truth frames as individual .bmp files.
- results/: An empty folder intended for saving the algorithm's binary output images.
- ROI.bmp / ROI.jpg: Image files defining the spatial Region of Interest (ROI) for evaluation.
- temporalROI.txt: A text file specifying the start and end frame numbers for scoring.

Groundtruth Labels

The groundtruth images use specific pixel values to label different regions:

- 0: Static (Represents the background).
- 50: Hard shadow.
- 85: Outside region of interest.
- 170: Unknown motion (e.g., motion blur, semi-transparency).
- 255: Motion (Represents the foreground).

4 Easy Problem

4.1 Overview and Objectives

The dataset provides a sequence of image frames for each video. The algorithm builds a background model over time and detects foreground pixels by comparing new frames with this model.

Two algorithms are implemented:

1. Mean Filter–based Background Subtraction
2. Median Filter–based Background Subtraction

Both are *temporal filtering* techniques that use pixel-wise intensity statistics to model the background.

4.2 Problem Definition

Given a sequence of image frames $I_t(x, y)$, where (x, y) are pixel coordinates and t denotes the frame index, we want to decompose each frame into:

$$I_t(x, y) = B_t(x, y) + F_t(x, y)$$

This equation expresses that each video frame is a combination of the static background and the moving foreground.

where:

- $B_t(x, y)$: Background (slowly changing or static components)
- $F_t(x, y)$: Foreground (moving objects)

The goal is to estimate $B_t(x, y)$ adaptively as frames evolve, while maintaining robustness against illumination changes and camera noise.

4.3 Algorithmic Pipeline

Both algorithms follow the same overall computational pipeline:

1. **Frame Acquisition:** The system scans the directory for all image files but utilizes a custom frame preprocessor module to selectively acquire only the relevant subset of frames (defined by start, stop, and step parameters) instead of reading the entire sequence.

2. **Preprocessing:** Convert each frame to grayscale to reduce dimensionality. For the Mean Filter specifically, pixel values are cast to floating-point precision (float64) to ensure accurate statistical variance calculations.
3. **Background Modeling:** Initialize the background using the first frame. For subsequent frames, perform a selective update: only pixels classified as 'background' are used to update the running mean/median, preventing moving objects from corrupting the model.
4. **Foreground Segmentation:** Apply a threshold-based decision to detect moving pixels.
5. **Noise Removal:** Apply morphological operations (erosion followed by dilation) to remove small noise blobs.
6. **Evaluation:** Compare detected foreground masks against ground-truth images to compute performance metrics.

4.4 Mean Filter Algorithm

4.4.1 Concept

Each pixel is modeled as a **Gaussian random variable** whose mean and variance evolve over time.

For each pixel location (x, y):

- Mean $\mu_t(x, y)$: estimate of background intensity
- Variance $\sigma_t^2(x, y)$: measure of pixel intensity fluctuation

4.4.2 Mathematical Formulation

The mean and variance are updated recursively using exponential smoothing:

$$\mu_t(x, y) = (1 - \alpha)\mu_{t-1}(x, y) + \alpha I_t(x, y)$$

$$\sigma_t^2(x, y) = (1 - \alpha)\sigma_{t-1}^2(x, y) + \alpha(I_t(x, y) - \mu_{t-1}(x, y))^2$$

where:

- α: learning rate (between 0 and 1)
- $(1 - \alpha)$: memory factor controlling adaptation speed

The recursive update rule is derived from the concept of Exponentially Weighted Moving Average (EWMA). Unlike a simple average which weights all past frames equally, EWMA assigns exponentially decreasing weights to older observations. The formula $\mu_t = (1 - \alpha)\mu_{t-1} + \alpha I_t$ can be expanded as:

$$\mu_t = \alpha I_t + \alpha(1 - \alpha)I_{t-1} + \alpha(1 - \alpha)^2 I_{t-2} + \dots$$

This expansion proves that the influence of older frames decays geometrically, allowing the model

to adapt to gradual lighting changes while forgetting obsolete history.

4.4.3 Foreground Detection

A pixel is classified as foreground if its intensity deviates significantly from the background model:

$$\textbf{Foreground if: } \frac{|I_t(x,y) - \mu_t(x,y)|}{\sigma_t(x,y)} \geq T$$

where T is a threshold (e.g., 3.0). This is equivalent to computing a normalized **Z-score** for each pixel to measure deviation strength.

4.4.4 Noise Removal and Postprocessing

After thresholding, a binary mask is obtained:

$$F_t(x,y) = \begin{cases} 255, & \text{if } \frac{|I_t(x,y)-\mu_t(x,y)|}{\sigma_t(x,y)} \geq T \\ 0, & \text{otherwise} \end{cases}$$

To refine results:

- **Erosion:** Removes isolated pixels or small noise blobs.
- **Dilation:** Restores the shape of valid detected objects.

Both are applied using a 3×3 kernel.

4.4.5 Optimization View

The mean filter corresponds to minimizing the **Mean Squared Error (MSE)** between the current pixel intensity and the estimated background:

$$\min_{\mu_t} \sum_t (I_t(x,y) - \mu_t(x,y))^2$$

This is a **convex, unconstrained optimization problem** because the squared loss is convex and differentiable. The running average thus provides the closed-form optimal solution.

4.5 Median Filter Algorithm

4.5.1 Concept

Instead of using the mean, this method estimates the **median intensity** at each pixel position over time. The median is inherently robust to short-term disturbances.

Intuitively:

- If an object appears briefly, its intensity is seen only for a few frames.

- The median ignores these short-term outliers, preserving the static background.

4.5.2 Mathematical Formulation

The background intensity is ideally the median of all past intensities:

$$B_t(x, y) = \text{median}\{I_1(x, y), I_2(x, y), \ldots, I_t(x, y)\}$$

To make computation efficient, an **approximate median update rule** is used:

$$B_t(x, y) = \begin{cases} B_{t-1}(x, y) + learning_rate, & \text{if } I_t(x, y) > B_{t-1}(x, y) \\ B_{t-1}(x, y) - learning_rate, & \text{if } I_t(x, y) < B_{t-1}(x, y) \\ B_{t-1}(x, y), & \text{otherwise} \end{cases}$$

This allows $B_t(x, y)$ to slowly approach the true median over time.

4.5.3 Foreground Detection

Foreground pixels are detected if their intensity difference from the estimated background exceeds a threshold:

$$|I_t(x, y) - B_t(x, y)| > \tau$$

where τ is a fixed threshold (e.g., 20).

4.5.4 Noise Removal and Postprocessing

As with the mean filter, erosion and dilation operations are applied to suppress noise and refine the detected regions. A 5x5 kernel is used.

4.5.5 Optimization View

The median filter minimizes the **L1 loss** between the observed and background intensities:

$$\min_{B_t} \sum_t |I_t(x, y) - B_t(x, y)|$$

This is also a **convex, unconstrained optimization problem**. Although the L1 norm is non-differentiable at zero, it still yields robust solutions against outliers.

4.6 Evaluation Methodology for Both Algorithms

After processing each frame, a binary mask of detected foreground is generated and stored. For quantitative evaluation, each output mask is compared with the ground-truth segmentation provided in the dataset.

- The metrics computed include:
  - True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN)
  - Precision, Recall, IoU and F-measure
- This enables an objective comparison of mean and median filtering performance.

4.7 Comparison: Mean vs Median Filters

Property	**Mean Filter**	**Median Filter**
Background model	Gaussian (mean + variance)	Non-parametric (median)
Loss minimized	$(I-\mu)^2$ (L2 loss)	$\lvert I-B\rvert$ (L1 loss)
Optimization type	Convex, unconstrained	Convex, unconstrained
Sensitivity	Sensitive to outliers	Robust to outliers
Update speed	Fast adaptation	Slower but stable
Use case	Dynamic background	Static scenes
Mathematical complexity	Moderate (variance update)	Low (simple increment/decrement)

4.8 Mathematical Glossary

Symbol	**Meaning**
$I_t(x,y)$	Pixel intensity at frame t
$\mu_t(x,y)$	Background mean (mean filter)
$\sigma_t^2(x,y)$	Background variance
$B_t(x,y)$	Background median estimate
α	Learning rate
T, τ	Thresholds for foreground detection
$F_t(x,y)$	Foreground mask
$Z_t = \frac{I_t(x,y)-\mu_t(x,y)}{\sigma_t(x,y)}$	Normalized Z-score difference
Kernel	$N \times N$ structuring element for noise removal

5 Hard Problem

5.1 Overview

While simple statistical filters perform well under steady conditions, they fail in complex or dynamic environments. The hard problem implements a more advanced background–foreground separation algorithm based on **Robust Principal Component Analysis (RPCA)**. RPCA provides a mathematically rigorous framework to achieve robust separation in such challenging cases.

The input video is treated as a sequence of grayscale frames stacked into a large data matrix:

$$M = [I_1, I_2, I_3, \ldots, I_T]$$

where each column (or row) corresponds to one frame flattened into a vector.

5.2 Problem Definition

The main objective of RPCA is to separate the observed video data M into:

$$M = L + S$$

where:

- L: Low-rank matrix representing the **background** (slowly changing, correlated pixels)
- S: Sparse matrix representing the **foreground** (moving objects or sudden changes)

Intuitively, L captures repetitive patterns across frames (background), while S captures transient or unusual elements (moving foreground).

5.3 Mathematical Formulation

The traditional RPCA can be described as the following optimization problem

$$\min_{L,S} \operatorname{rank}(L) + \lambda \|S\|_0 \quad \text{s.t. } M = L + S, \tag{5.1}$$

where $\operatorname{rank}(L)$ is the rank function of the matrix(number of non-zero singular values in L), $\|\cdot\|_0$ is the l_0 norm of the matrix (the number of non-zero elements in S), and λ is a regularization parameter that balances the trade-off between preferring a simpler (lower rank) L and preferring a sparser S. Larger λ penalizes nonzero entries in S more strongly (so favors explaining data with

L), smallerλ allows more sparsity in S. Because the rank function and l_0 norm are non-convex discrete functions, the solution of this model is NP-hard.

Why non-convex and discrete? rank$(\cdot)$ jumps in integer steps and is not convex. $\|\cdot\|_0$ is combinatorial — it requires choosing which individual entries are nonzero. Both create a search over combinatorial possibilities.

Why NP-hard? Minimizing combinations of rank and $\|\cdot\|_0$ is computationally infeasible for moderate sizes in the worst case. There's no known algorithm that runs in polynomial time that always finds the global optimum.

We need tractable approximations (convex relaxations) that can be solved reliably with polynomial-time algorithms. The nuclear function and l_1 norm of the matrix can be used to convexly approximate the rank function and l_0 norm of the matrix, respectively. Convex problems have no local minima (only global minima) and can be solved efficiently with well-understood algorithms.

Hence, the above RPCA model can be approximated as a convex optimization problem known as **Principal Component Pursuit (PCP)** The objective is to recover L and S by minimizing their complexity while keeping their sum equal to the observed data matrix:

$$\min_{L,S} \|L\|_*+\lambda\|S\|_1 \quad \text{s.t.} \quad M = L + S$$

where:

- $\|L\|_*= \sum_i \sigma_i(L)$: Nuclear norm (sum of singular values).Minimizing nuclear norm encourages many singular values to shrink toward zero, promoting low-rank solutions.
- $\|S\|_1= \sum_{i,j}|S_{ij}|$: Sum of values of the entries in the matrix. Minimizing L_1- norm tends to push many entries exactly to zero while allowing some nonzero entries.
- λ: Weighting parameter controlling the trade-off between low-rank and sparsity (typically $\lambda = 1/\sqrt{\max(m,n)}$)

This formulation ensures that background pixels (highly correlated across frames) are modeled as low-rank, while moving objects (appearing sparsely) form the sparse component.

5.4 Optimization Algorithm: Inexact Augmented Lagrange Multiplier (IALM)

The **Inexact Augmented Lagrange Multiplier (IALM)** method is preferred for its fast convergence and strong theoretical guarantees in convex problems involving low-rank and sparse components.

5.4.1 Derivation of the Augmented Lagrangian

We begin with the original Principal Component Pursuit (PCP) formulation:

$$\min_{L,S} \|L\|_*+\lambda\|S\|_1 \qquad \text{s.t. } M = L + S.$$

This is a constrained convex optimization problem. Because of the equality constraint, it cannot be solved directly using standard unconstrained methods. The classical Lagrangian is a mathematical tool used to convert a constrained optimization problem into an unconstrained one by introducing extra variables called Lagrange multipliers. The multiplier enforces the constraint during optimization.

If the constraint is satisfied g(x)=0, the Lagrangian does nothing extra.

If the constraint is violated g(x) = 0, the term λ g(x) increases the cost.

So minimizing the Lagrangian naturally pushes the solution back toward g(x)=0.

Step 1: Classical Lagrangian

For a constrained problem of the form

$$\text{s.t. } g(L, S) = 0,$$

the classical Lagrangian is given by

$$\mathcal{L}(L, S, Y) = f(L, S) + \langle Y,\ g(L, S)\rangle.$$

In our RPCA formulation:

$$f(L, S) = \|L\|_* + \lambda\|S\|_1,$$

$$g(L, S) = M - L - S = 0.$$

Thus, the classical Lagrangian becomes

$$\mathcal{L}(L, S, Y) = \|L\|_* + \lambda\|S\|_1 + \langle Y,\ M - L - S\rangle.$$

Need for Augmentation

Using only the dual variable Y to enforce the constraint typically leads to slow and unstable convergence. The constraint

$$M - L - S = 0$$

is enforced weakly in the classical Lagrangian, causing oscillations during optimization. This motivates the use of the **Augmented Lagrangian**, which stabilizes optimization by adding a quadratic penalty term.

Step 2: Adding the Quadratic Penalty

The Augmented Lagrangian adds the term

$$\frac{\mu}{2}\|M - L - S\|_F^2,$$

which penalizes violation of the equality constraint.

This term forces

$$M - L - S \approx 0$$

more strongly, not only through the dual variable Y, but also through the quadratic penalty.

Final Augmented Lagrangian Expression

Combining the objective, classical Lagrangian term, and quadratic penalty, we obtain the Augmented Lagrangian used in RPCA:

$$\mathcal{L}(L, S, Y, \mu) = \underbrace{\|L\|_* + \lambda\|S\|_1}_{\text{original objective}} + \underbrace{\langle Y,\ M - L - S\rangle}_{\text{constraint enforcement}} + \underbrace{\frac{\mu}{2}\|M - L - S\|_F^2}_{\text{quadratic penalty}}$$

Why This Formulation Works

Term	Purpose
$\|L\|_*$	Promotes low-rank background L
$\lambda\|S\|_1$	Promotes sparsity in the foreground S
$\langle Y, M - L - S\rangle$	Classical Lagrange multiplier enforcing the constraint
$\frac{\mu}{2}\|M - L - S\|_F^2$	Stabilizes optimization and accelerates convergence

Thus, the Inexact Augmented Lagrange Multiplier (IALM) method becomes an efficient and theoretically sound algorithm for solving the RPCA problem.

5.5 Inexact ALM Strategy

Solving for L and S simultaneously is too difficult. IALM solves them *alternately*:

1. **Update S:** Keep L and Y fixed. Find the best sparse foreground that explains whatever the background failed to capture.
2. **Update L:** Freeze S. Find the simplest (lowest-rank) background that explains the remaining structure.
3. **Update Y:** Measure the current error $M - L - S$ and record it so the next iteration fixes it.
4. **Increase μ:** As we get closer, we penalize mistakes more strongly, forcing the algorithm to zero in on the true background/foreground.

These four steps repeat until both primal and dual errors are extremely small, meaning the algorithm has effectively solved the RPCA problem.

5.6 Mathematical Description of the IALM Method

5.6.1 Algorithm Pipeline

1. **Input:**
   - Data matrix $M \in R^{m \times n}$ formed by stacking grayscale video frames.
   - Regularization parameter $\lambda = \frac{1}{\sqrt{\max(m,n)}}$.
2. **Initialization:**
   - $L_0 = 0, \quad S_0 = 0$
   - Dual variable:
   $$Y_0 = \frac{M}{\min(\|M\|_2,\ \|M\|_\infty)}$$
   - Penalty parameter:
   $$\mu_0 = \frac{1.25}{\|M\|_2}, \qquad \rho \in (1, 1.6]$$
   - Set iteration counter $k = 0$
3. **Iterative Updates (IALM):** For each iteration:
$$\begin{aligned} S_{k+1} &= \text{SoftThreshold}\left(M - L_k + \frac{1}{\mu_k} Y_k,\ \frac{\lambda}{\mu_k}\right), \\ L_{k+1} &= \text{SVT}\left(M - S_{k+1} + \frac{1}{\mu_k} Y_k,\ \frac{1}{\mu_k}\right), \\ Y_{k+1} &= Y_k + \mu_k \left(M - L_{k+1} - S_{k+1}\right). \end{aligned}$$
4. **Penalty Update:**
$$\mu_{k+1} = \min(\rho\, \mu_k,\ \mu_{\max})$$
5. **Stopping Criteria:** Stop when:
$$\frac{\|M - L_{k+1} - S_{k+1}\|_F}{\|M\|_F} < \epsilon_{\text{primal}}, \qquad \frac{\mu_k \|S_{k+1} - S_k\|_F}{\|M\|_F} < \epsilon_{\text{dual}}.$$
6. **Output:**
   - Low-rank background L
   - Sparse foreground S
   - Estimated rank:
   $$\text{rank}(L) = \#\{\sigma_i(L) > 0\}$$

5.6.2 Why Normalization of the Dual Variable is Necessary

In the Augmented Lagrangian formulation of RPCA, the dual variable Y is updated at each iteration according to:

$$Y^{(k+1)} = Y^{(k)} + \mu^{(k)} \left(M - L^{(k+1)} - S^{(k+1)} \right),$$

which means that the behavior of the algorithm is highly sensitive to the scale of the initial multiplier Y_0. If Y_0 is too large, the dual ascent step overshoots, causing numerical instability. If Y_0 is too small, the constraint enforcement becomes weak and convergence slows down significantly. To guarantee stability, RPCA literature recommends initializing the dual variable such that its magnitude is at most one. This is achieved by normalizing the data matrix M using the quantity

$$J = \min(\|M\|_2,\ \|M\|_\infty),$$

where $\|M\|_2$ is the spectral norm (largest singular value) and $\|M\|_\infty = \max_{i,j}|M_{ij}|$ is the maximum absolute entry.

The normalized dual variable is then defined as:

$$Y_0 = \frac{M}{J},$$

which guarantees

$$\|Y_0\|_2 \leq 1 \quad \text{and} \quad \|Y_0\|_\infty \leq 1.$$

5.6.3 Update Rules Explained

Sparse Update (Soft Thresholding):

$$S^{(k+1)} = \text{SoftThreshold} \left(M - L^{(k)} + \frac{1}{\mu^{(k)}} Y^{(k)},\ \frac{\lambda}{\mu^{(k)}} \right)$$

The sparse update uses the current reconstruction error $M - L^{(k)}$ together with the dual correction term $Y^{(k)}/\mu^{(k)}$. Soft-thresholding is then applied, which removes small values (noise) while preserving large values corresponding to moving objects. The threshold $\lambda/\mu^{(k)}$ decreases as iterations proceed, making the algorithm increasingly precise.

Low-Rank Update (SVT):

$$L^{(k+1)} = U\, \text{diag}(\text{SoftThreshold}(s,\ 1/\mu^{(k)}))\, V^T$$

where U, s, V are from the SVD of the current estimate.

Dual and Penalty Updates

We update the dual variable (Lagrange multiplier) and the penalty parameter as:

$$Y^{(k+1)} = Y^{(k)} + \mu^{(k)}(M - L^{(k+1)} - S^{(k+1)}), \qquad \mu^{(k+1)} = \rho\,\mu^{(k)}.$$

The Y-update is a dual-ascent step on the augmented Lagrangian and accumulates the constraint residual. Increasing μ tightens the quadratic penalty $\frac{\mu}{2}\|M - L - S\|_F^2$, which reduces the SVT and soft-threshold parameters ($1/\mu$ and λ/μ) and thus moves the solver from coarse, robust corrections toward fine, precise enforcement of $M = L + S$.

5.6.4 Stopping Conditions

$$\frac{\|M - L - S\|_F}{\|M\|_F} < \epsilon_{\text{primal}}, \qquad \frac{\mu_{\text{k}}\|S^{(k)} - S^{(k-1)}\|_F}{\|M\|_F} < \epsilon_{\text{dual}}.$$

Primal error: Measures how well the constraint is satisfied

Dual error: Measures whether the sparse component is stabilizing

Typical thresholds:

$$\epsilon_{\text{primal}} = 10^{-7}, \qquad \epsilon_{\text{dual}} = 10^{-5}$$

6 Results

6.1 Scores Description

— **Precision** (The Noise Score): Question it answers: Of all the white pixels your algorithm created, what percentage of them were correct (i.e., actually a car)?

— **Recall** (The Finding Score) Question it answers: Of all the real car pixels (from the ground truth), what percentage did your algorithm successfully find?

— **F1-Score** (The Balanced Average): Question it answers: What is the balanced average of Precision and Recall?

— **IoU** (Intersection over Union) Question it answers: How much did your detected white shape overlap with the real white shape?

— **Ideal scenario:** For all four of those scores, 1.0 is a perfect score.

Precision = 1.0 means we have zero noise (no false positives).

Recall = 1.0 means we found 100% of the real objects (no false negatives).

F1-Score = 1.0 means we have perfect Precision and perfect Recall.

IoU = 1.0 means our detected shapes match the real shapes perfectly.

6.2 Baseline - Highway Results

Table 6.1: Comparison of Performance

Algorithm	**Precision**	**Recall**	**F1-Score**	**IoU**	**Exec. Time (s)**
Mean Filter	[0.6712]	[0.9031]	[0.7701]	[0.6261]	[83.93s]
Median Filter	[0.6753]	[0.6130]	[0.6426]	[0.4734]	[39.60s]
RPCA	[0.8137]	[0.8289]	[0.8212]	[0.6967]	[285.16s]

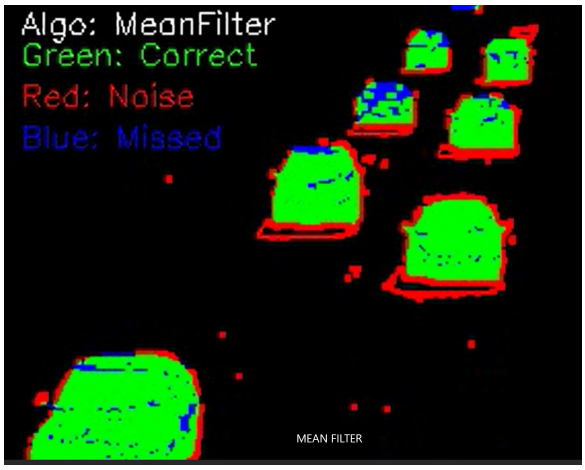
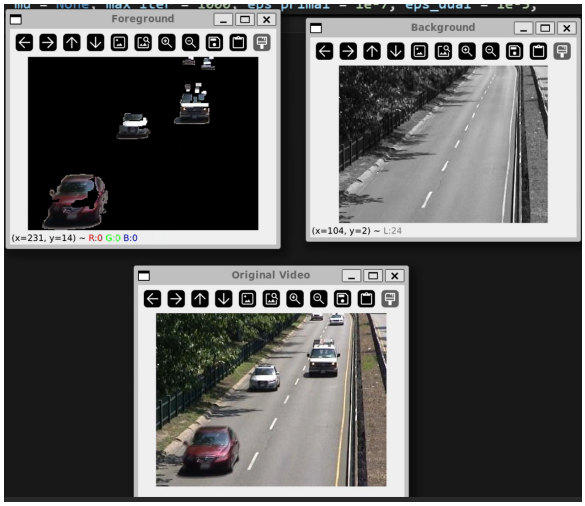


Figure 6.1: Visualisation of the Mean Filter results (Image 1 and Image 2).



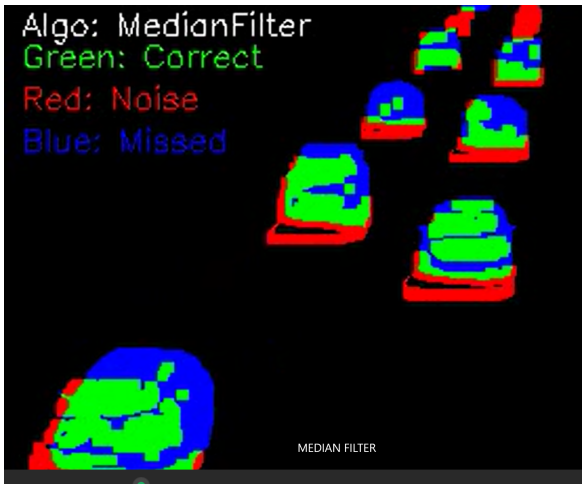
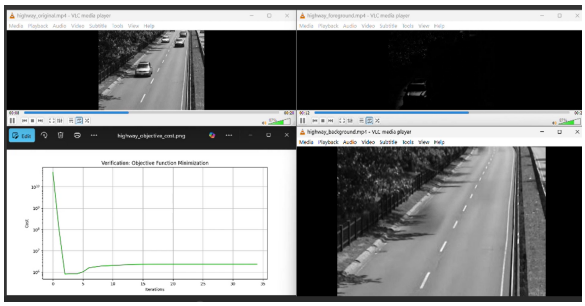


Figure 6.2: Visualisation of the Median Filter results (Image 1 and Image 2).



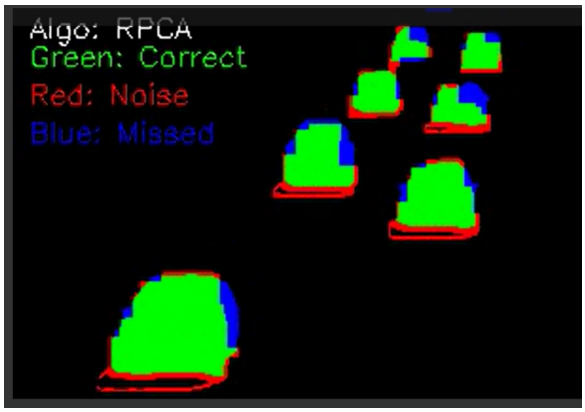


Figure 6.3: Visualisation of the RPCA results.

RPCA Analysis:

Final Objective Value: 2322447.5288

1. Reconstruction Error (Relative): 6.69e-08

Mathematical decomposition holds (M approx L + S).

2. Foreground Sparsity: 18.31%

3. Background Rank: 279

The following analysis compares performance between the "Easy" and "Hard" scenarios based on the metrics:

```
Gain in Accuracy: +17.86%
Cost in Time:     +245.57 seconds
Price of Realism: 13.75 seconds per 1% accuracy gain
```

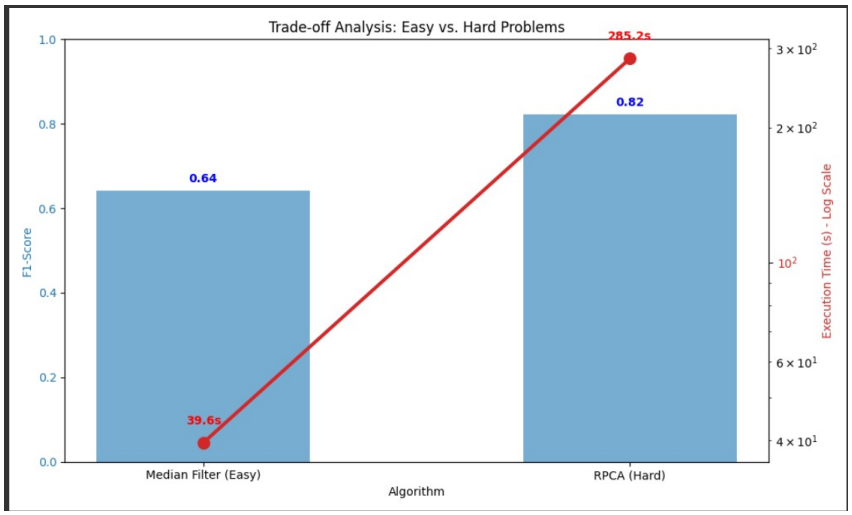
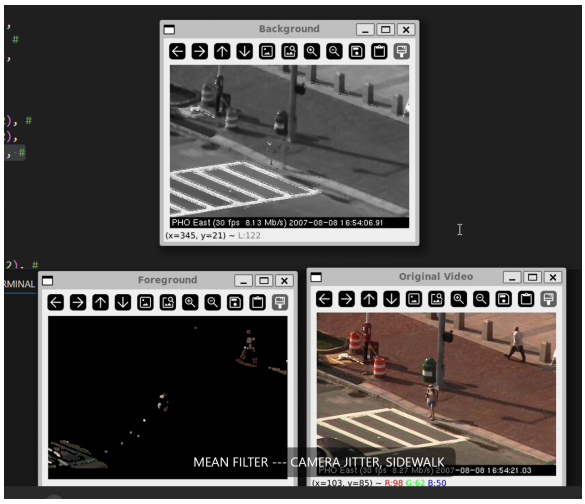


Figure 6.4: Plot illustrating the comparison metrics (e.g., accuracy vs. time) for Easy and Hard scenarios.

6.3 CameraJitter - Sidewalk Results

Table 6.2: Comparison of Performance

Algorithm	Precision	Recall	F1-Score	IoU	Exec. Time (s)
Mean Filter	[0.1863]	[0.3918]	[0.2525]	[0.1445]	[46.00s]
Median Filter	[0.3938]	[0.3086]	[0.3461]	[0.2092]	[40.76s]
RPCA	[0.8281]	[0.4309]	[0.5668]	[0.3955]	[112.68s]



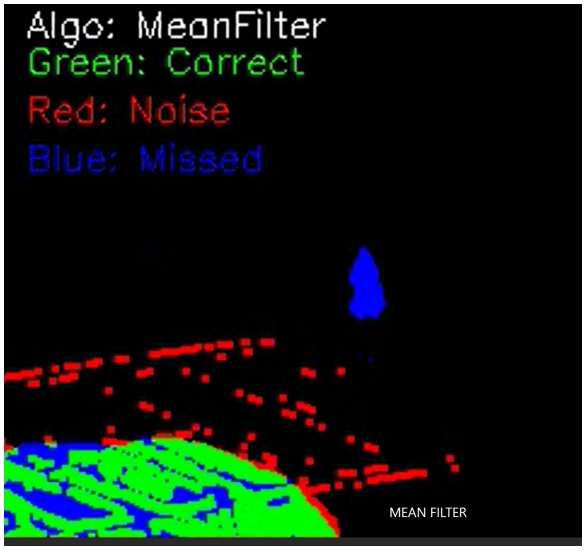
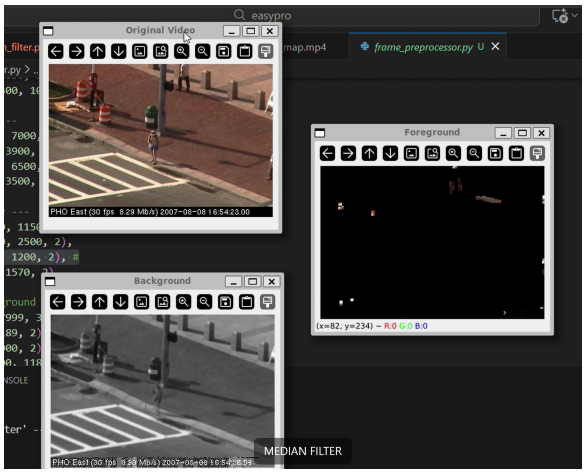


Figure 6.5: Visualisation of the Mean Filter results (Image 1 and Image 2).



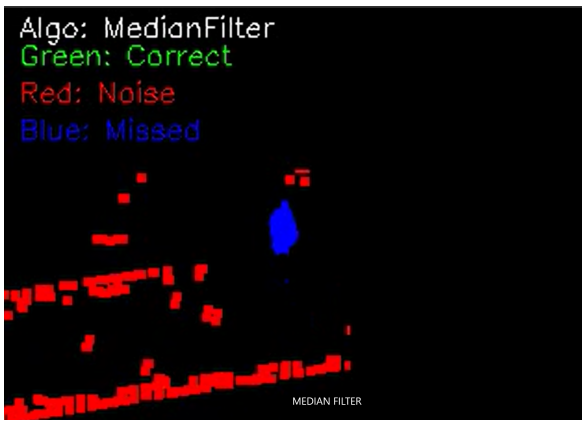
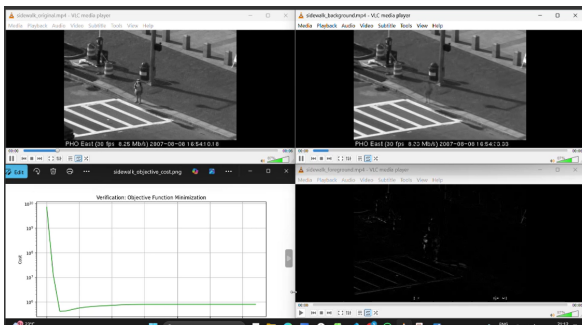


Figure 6.6: Visualisation of the Median Filter results (Image 1 and Image 2).



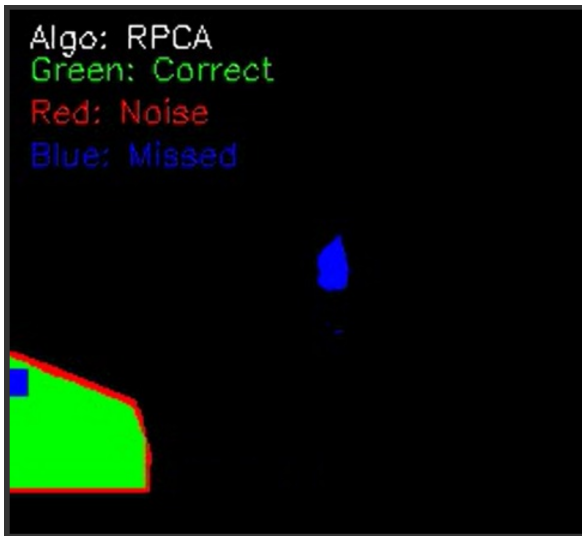


Figure 6.7: Visualisation of the RPCA results.

RPCA Analysis:

Final Objective Value: 795013.0045

1. Reconstruction Error (Relative): 9.33e-08

Mathematical decomposition holds (M approx L + S).

2. Foreground Sparsity: 16.95%

3. Background Rank: 51

The following analysis compares performance between the "Easy" and "Hard" scenarios based on the loaded data:

```
Gain in Accuracy: +22.08%
Cost in Time:     +71.92 seconds
Price of Realism: 3.26 seconds per 1% accuracy gain
```

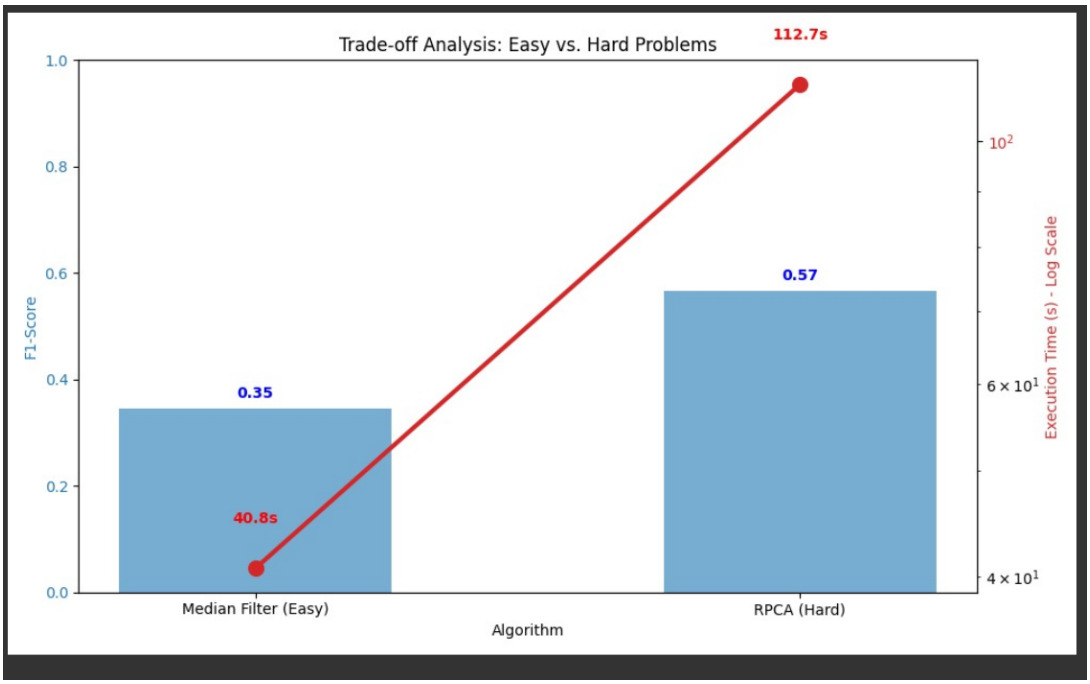
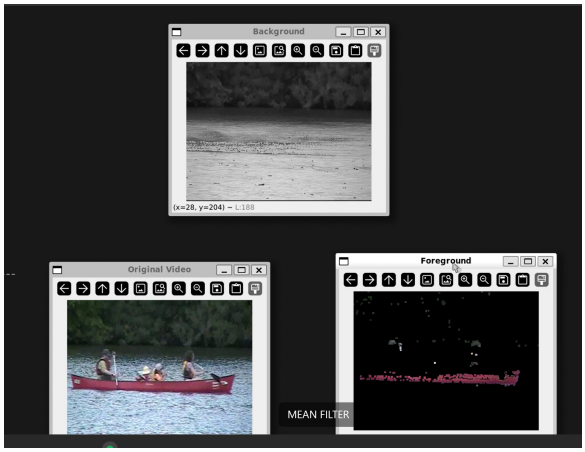


Figure 6.8: Plot illustrating the comparison metrics (e.g., accuracy vs. time) for Easy and Hard scenarios.

6.4 DynamicBackground - Canoe Results

Table 6.3: Comparison of Performance

Algorithm	Precision	Recall	F1-Score	IoU	Exec. Time (s)
Mean Filter	[0.5075]	[0.3654]	[0.4249]	[0.2698]	[23.93s]
Median Filter	[0.6729]	[0.7035]	[0.6879]	[0.5243]	[22.47s]
RPCA	[0.7275]	[0.3704]	[0.4909]	[0.3253]	[113.89s]



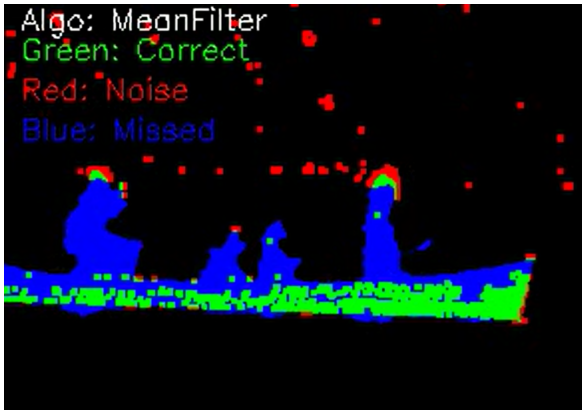
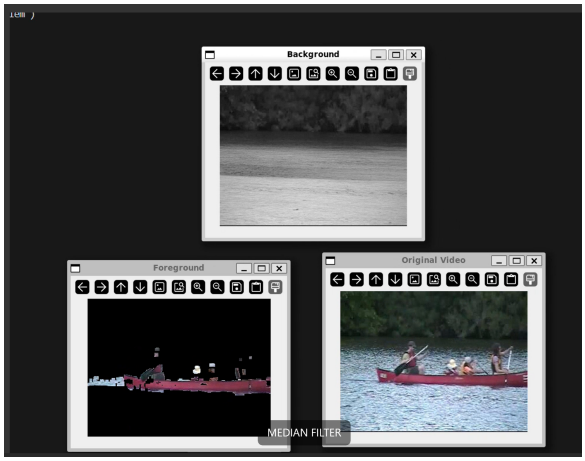


Figure 6.9: Visualisation of the Mean Filter results (Image 1 and Image 2).



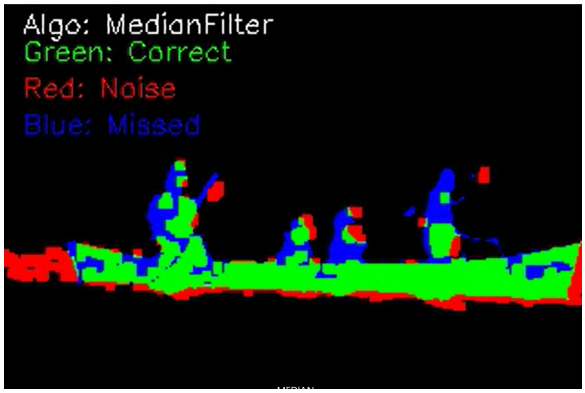
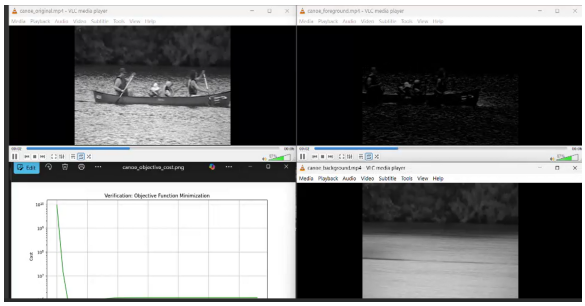


Figure 6.10: Visualisation of the Median Filter results (Image 1 and Image 2).



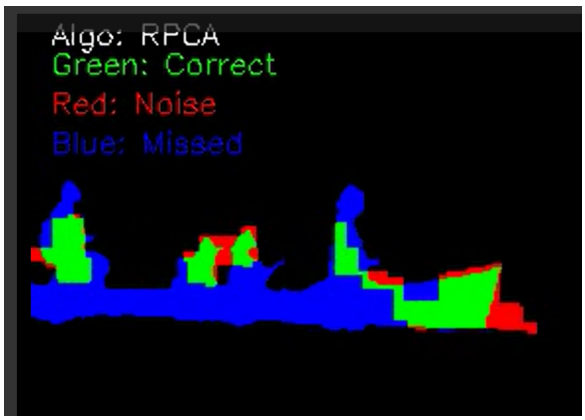


Figure 6.11: Visualisation of the RPCA results.

RPCA Analysis:

Final Objective Value: 1211823.9560

1. Reconstruction Error (Relative): 6.02e-08

Mathematical decomposition holds (M approx L + S).

2. Foreground Sparsity: 47.59%

3. Background Rank: 106

The following analysis compares performance between the "Easy" and "Hard" scenarios based on the loaded data:

```
Gain in Accuracy: +19.70%
Cost in Time:     +91.42 seconds
Price of Realism: 0.00 seconds per 1% accuracy gain
```

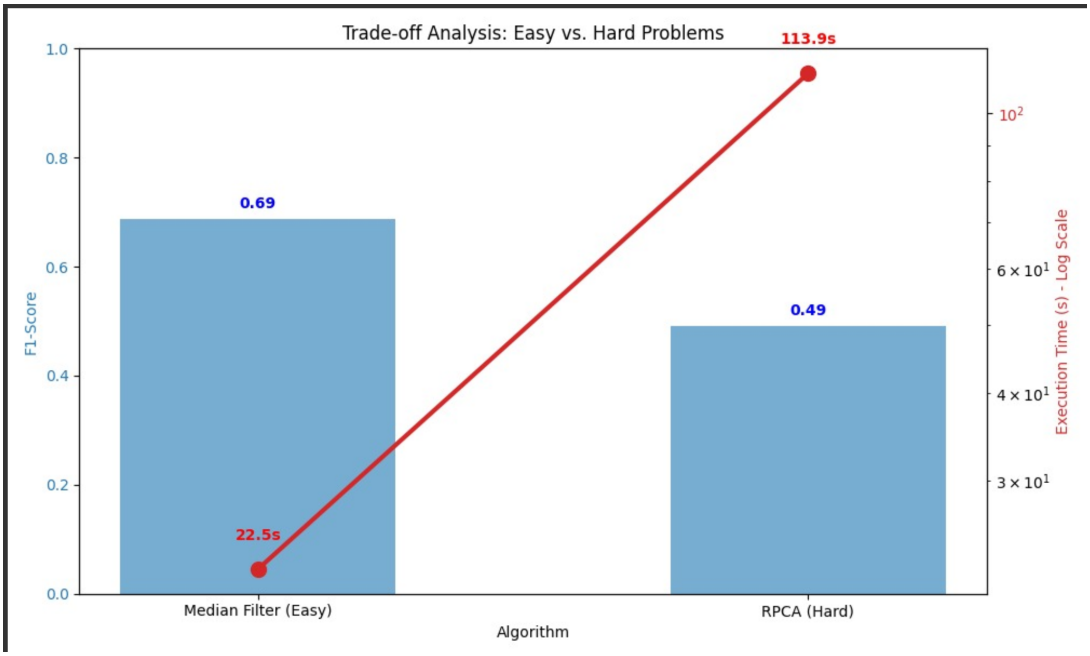
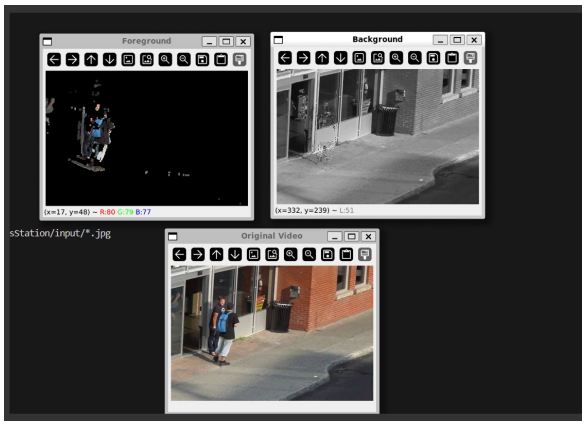


Figure 6.12: Plot illustrating the comparison metrics (e.g., accuracy vs. time) for Easy and Hard scenarios.

6.5 Shadow - BusStation Results

Table 6.4: Comparison of Performance

Algorithm	Precision	Recall	F1-Score	IoU	Exec. Time (s)
Mean Filter	[0.5679]	[0.6857]	[0.6213]	[0.4506]	[54.90s]
Median Filter	[0.5553]	[0.3996]	[0.4648]	[0.3027]	[42.28s]
RPCA	[0.6890]	[0.3970]	[0.5038]	[0.3367]	[220.42s]



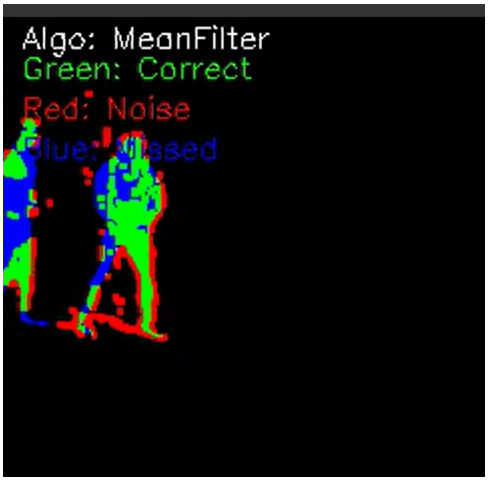
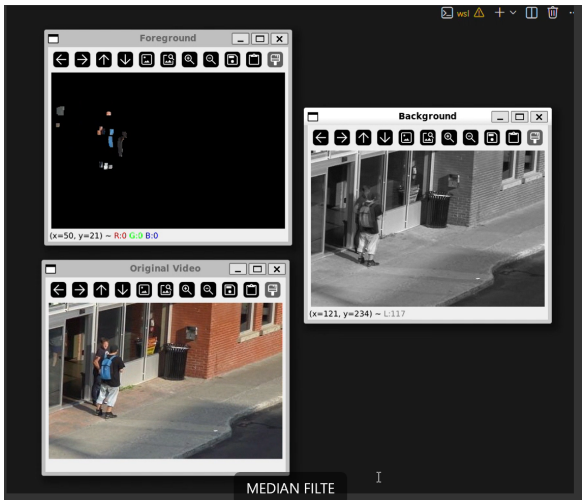


Figure 6.13: Visualisation of the Mean Filter results (Image 1 and Image 2).



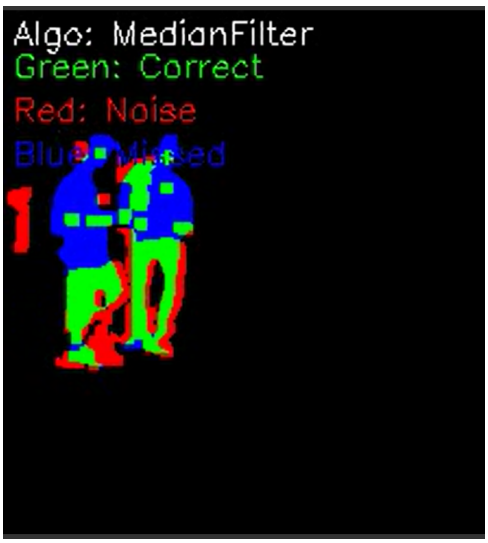
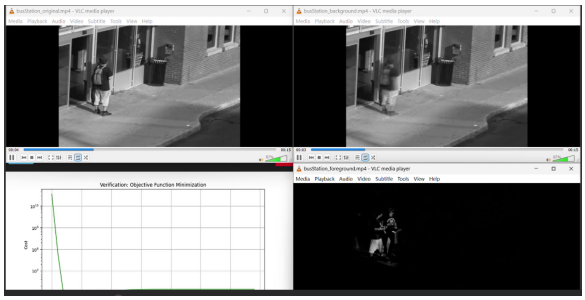


Figure 6.14: Visualisation of the Median Filter results (Image 1 and Image 2).



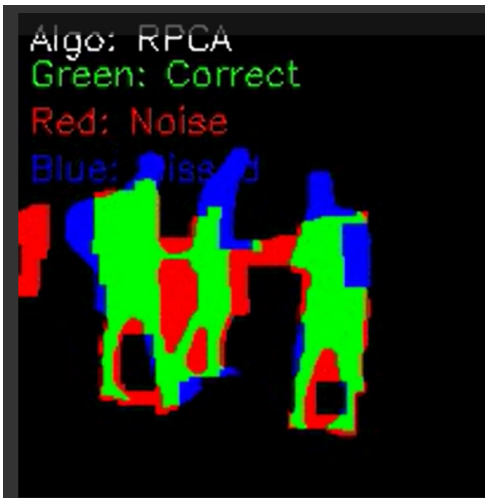


Figure 6.15: Visualisation of the RPCA results.

RPCA Analysis:

Final Objective Value: 1393204.6244

1. Reconstruction Error (Relative): 6.41e-08

Mathematical decomposition holds (M approx L + S).

2. Foreground Sparsity: 7.54%

3. Background Rank: 185

The following analysis compares performance between the "Easy" and "Hard" scenarios based on the loaded data:

```
Gain in Accuracy: +3.90%
Cost in Time:     +178.13 seconds
Price of Realism: 45.69 seconds per 1% accuracy gain
```

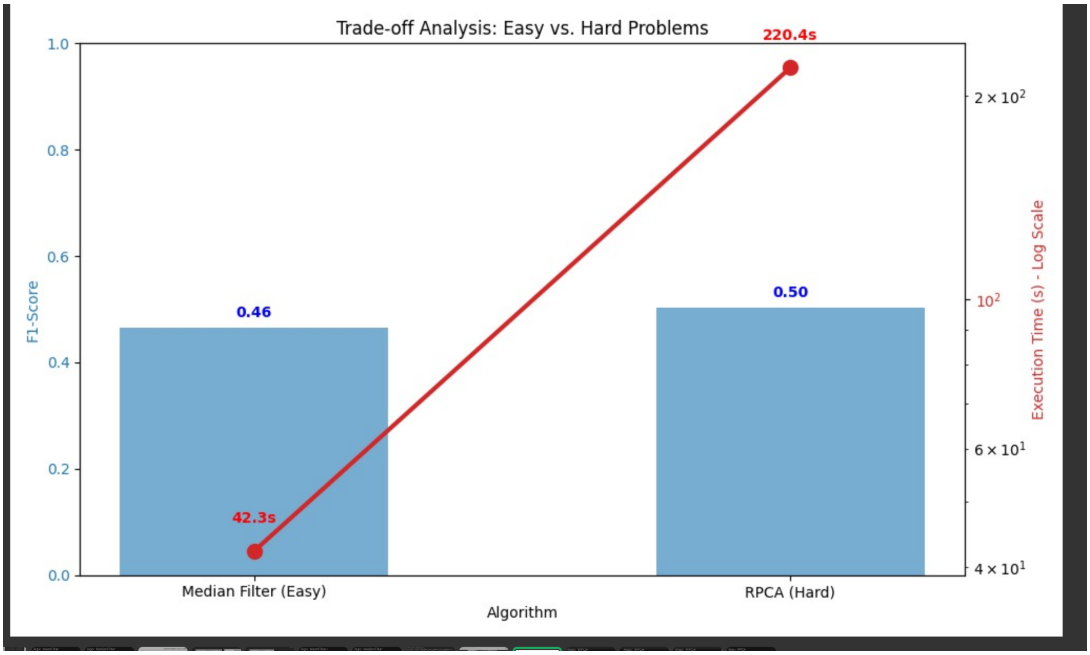


Figure 6.16: Plot illustrating the comparison metrics (e.g., accuracy vs. time) for Easy and Hard scenarios.

6.6 Comparative Analysis

Easy Problem:
In the easy problem, background separation is achieved using simple pixel-wise statistical methods such as mean or median filtering. These models operate on each frame independently using local temporal or spatial statistics and involve simple convex operations. For a video with N frames and P pixels per frame:

$$\text{Per-frame cost: } O(P), \qquad \text{Total cost: } \mathbf{O(N \cdot P)}.$$

For the median filter, if the kernel size is fixed, the per-pixel work remains constant; hence the overall complexity remains $O(N \cdot P)$.

Hard Problem:
The hard problem uses a **global matrix-level optimization** approach such as Robust PCA (RPCA). The entire video is arranged as a matrix $M \in R^{N \times P}$ and decomposed into a low-rank background L and sparse foreground S. This captures spatial and temporal correlations but requires repeated matrix decompositions (SVD-like steps). The dominant per-iteration computational cost is:

$$\text{Per-iteration cost: } O(N \cdot P \cdot k),$$

where k is the number of singular values/vectors computed. After I iterations, the total computational cost becomes:

$$\text{Total cost: } \mathbf{O(I \cdot N \cdot P \cdot k)}.$$

Thus, RPCA adds a significant **layer of complexity**: it transforms local temporal filtering into a global convex optimization problem.

The result is a model that is slower but much more accurate and robust in challenging, real-world video environments.

6.7 The Price of Realism

The superior performance of RPCA comes at a specific cost, which we define as the **Price of Realism**. This is the cost incurred when forcing a mathematical model to adapt to the messy, stochastic nature of real-world data. We observed this price in two distinct forms:

Type A: The Computational Price (Efficiency vs. Accuracy) Realism requires processing power. The RPCA algorithm operates at a much higher computational cost .To achieve the observed gain in F1-Score, we sacrificed significant computational resources, effectively paying an "exchange rate" of computation time for every percentage point improvement in accuracy.

Type B: The Theoretical Price (Rank Inflation) Theoretically, a static highway background should have a very low rank. However, our optimal solution converged at a significantly higher rank. This violation of the low-rank assumption represents the theoretical price of realism: real-world factors such as camera jitter, sensor noise, swaying trees, and subtle lighting shifts forced the optimization algorithm to inflate the rank of the background matrix L to absorb these variations. To fit the reality of the dataset, the algorithm had to construct a mathematical model more complex than theory predicted.

6.8 Visualizing Key Trade-Offs

The visual difference maps and quantitative metrics have been presented above.

6.8.1 Analysis for highway:

— **Median Filter** acts as an overly conservative gatekeeper; with the lowest Recall (0.6130), it fails to detect nearly 40% of moving objects, likely missing low-contrast vehicles. — **Mean Filter** achieves the high Recall, successfully capturing a lot of motion event. However, this comes at the cost of the lowest Precision (0.6712). The algorithm suffers from significant ghosting artifacts—where the trailing edges of moving cars are incorrectly flagged as foreground—making it noisy and unreliable for precise segmentation.

— **RPCA** demonstrates superior robustness. It achieves a balanced high performance, securing both the highest Precision (0.8137) and a strong Recall (0.8289). Unlike the Mean Filter, which aggressively captures motion (Recall 0.90) at the cost of significant noise (Precision 0.67), RPCA successfully isolates true foreground objects while rejecting environmental noise.It also achieved highest IoU(0.6967)

— **Execution Time metrics** highlights a hard barrier in applicability.

The Median Filter (39.60s) and Mean Filter (83.93s) operate with low computational overhead, making them viable for online, real-time surveillance.

RPCA (285.16s), however, is approximately 7x slower than the Median filter. As a batch algorithm requiring Iterative SVD, it demands the full video duration to be loaded into memory. This restricts its superior accuracy (F1-Score 0.82) to where processing speed is secondary to quality.

— **RPCA** is adaptive. By allowing the background rank to inflate, it mathematically absorbed the environmental chaos and camera jitter into the Low-Rank matrix. This adaptability is exactly why it achieved the highest IoU (0.6967), effectively paying a computational price to stabilize the background model against real-world instability.