HOMEWORK 4B

ECE/CS 8690 2302 Computer Vision

Moving Object Detection using Simple Background Subtraction

Xuanbo Miao

14422044

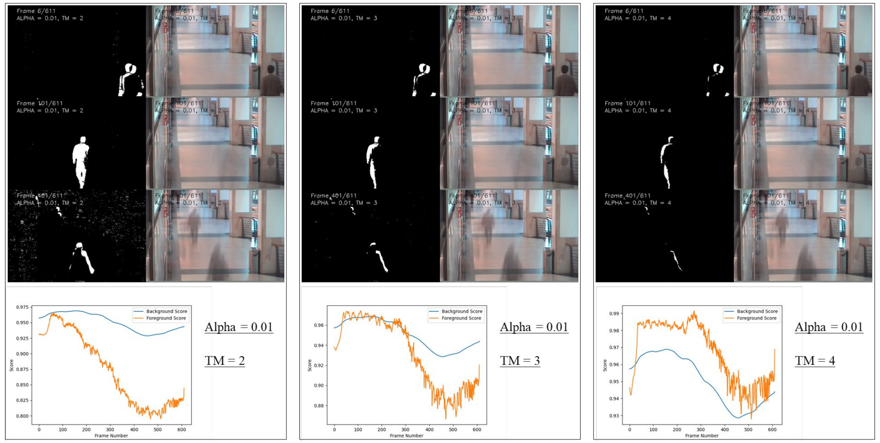
xmiao@mail.missouri.edu

Abstract

This assignment focuses on implementing a basic pipeline for feature detection, description, and matching. The first step is feature detection, which involves computing the autocorrelation matrix and a scalar feature detection measure using a formula discussed in class. The detected feature points are then shown in the image and included in the report. The second step is feature matching and evaluation, which involves defining and describing a feature descriptor and using it to match features using three different strategies. The performance of the feature matching is evaluated using transformation/homography matrices provided with the data.

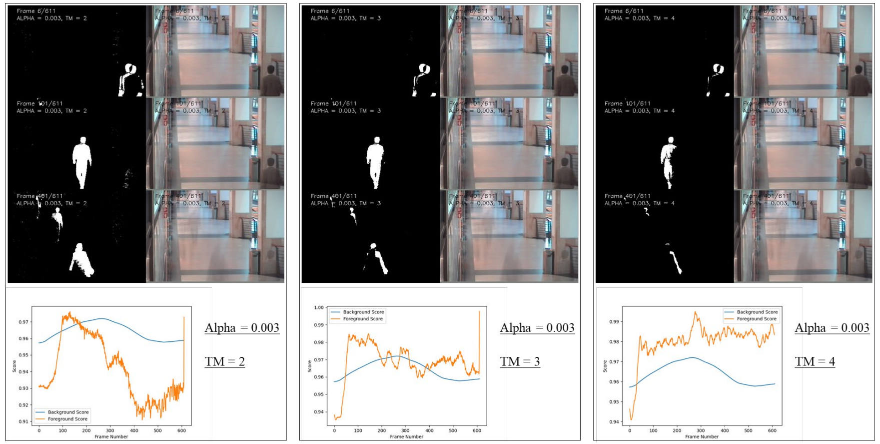
Alpha = 0.01

①



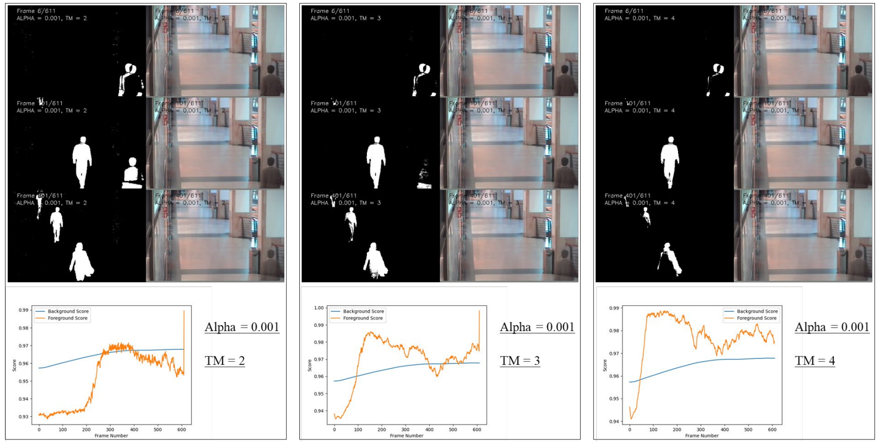
Alpha = 0.003

②



Alpha = 0.001

③



Introduction

In the field of computer vision, detecting and tracking moving objects in video streams is a crucial task with numerous applications, including surveillance, traffic monitoring, and robotic navigation. One popular technique for accomplishing this is through background subtraction, where the objective is to identify the foreground, consisting of moving objects, by distinguishing it from the static background.

In this assignment, we will implement a simple background subtraction algorithm using a single Gaussian model for each pixel to detect moving objects in a video sequence captured by a stationary camera. The algorithm involves maintaining a background model, classifying pixels as foreground or background based on their differences from the background model, and updating the background model with new frames over time.

We will experiment with different learning rates and matching thresholds to analyze the performance of the algorithm in detecting moving objects. The test data for this assignment is taken from the CAVIAR1 dataset, and the results will be evaluated using frames 5, 100, and 400. The assignment will be assessed on the basis of the source code, visualizations, and interpretations of the results.

Moving Object Detection using Simple Background Subtraction

The goal of this assignment is to implement a very simple background subtraction algorithm to detect moving objects in a scene imaged using a stationary camera. In background subtraction methods, moving regions are detected through difference between the current frame and a reference background image.

$$|\,frame_i - Background_i\,| > Th$$

These approaches provide the most complete feature data but are often sensitive to dynamic scene changes due to lighting and extraneous events.

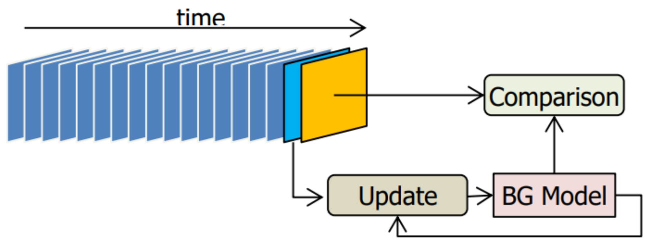
<u>Assignment:</u>

Implement a very simple background subtraction algorithm where previous values for each pixel is modeled using a single Gaussian distribution (μ,σ).

Mean and Covariance (Single Gaussian)

Update equations:

- $\mu(i,j,t+1) = \alpha\, frame(i,j,t) + (1-\alpha)\mu(i,j,t)$
- $\sigma^2(i,j,t+1) = \alpha(frame(i,j,t) - \mu(i,j,t))^2 + (1-\alpha)\sigma^2(i,j,t)$



Codes as required

```
class BGSubModel:

    # The model is initialized with the first frame.
    def __init__(self, first_frame, alpha, tm, true_bg):
        self.mean = np.float32(first_frame)
        self.var = np.ones_like(self.mean) * 128
        self.alpha = alpha
        self.tm = tm
        self.true_bg = np.float32(true_bg)

    # Classify the current frame as foreground or background
    def classify(self, current_frame):
        diff = np.abs(np.float32(current_frame) - self.mean)
        self.fg_mask = np.where(
            diff > (self.tm * np.sqrt(self.var)), 255, 0).astype(np.uint8)

        diff_true = np.abs(np.float32(current_frame) - self.true_bg)
        self.true_fg_mask = np.where(
            diff_true > (self.tm * np.sqrt(self.var)), 255, 0).astype(np.uint8)

        return self.fg_mask

    def evaluate(self):
        mse_bg = np.mean((self.true_bg - self.mean) ** 2)
        max_val_bg = np.max((self.true_bg.max(), self.mean.max()))
        score_bg = 1 - np.sqrt((mse_bg / (max_val_bg ** 2)))

        mse_fg = np.mean((self.true_fg_mask - self.fg_mask) ** 2)
        max_val_fg = np.max((self.true_fg_mask.max(), self.fg_mask.max()))
        # considering fg is only a very small part of the image, so we add a
scale factor to make the score more reasonable
        factor = 100
        score_fg = 1 - np.sqrt((mse_fg / (max_val_fg ** 2)))*factor

        return score_bg, score_fg

    # Update the model with the current frame
    def update(self, current_frame):
        inv_alpha = 1 - self.alpha
        self.mean = inv_alpha * self.mean + \
            self.alpha * np.float32(current_frame)
        self.var = inv_alpha * self.var + \
            self.alpha * (np.float32(current_frame) - self.mean) ** 2
```

```
def main():
    if not os.path.isdir(OUTPUT_PATH):
        os.mkdir(OUTPUT_PATH)
    flist = [f for f in os.listdir(INPUT_PATH) if f.endswith('.png')]
    flist = sorted(flist)
    n = len(flist)

    true_bg = cv2.imread(BG_true_PATH)

    for ALPHA in ALPHA_list:
        for TM in TM_list:
            # print the parameters
            print(f'ALPHA = {ALPHA}, TM = {TM}')
            # Read the first image and initialize the model
            im = cv2.imread(os.path.join(INPUT_PATH, flist[0]))
            bg_model = BGSubModel(im, ALPHA, TM, true_bg)

            # Set up the VideoWriter objects
            # the name should contain the parameters of the model, ALPHA and TM
            fourcc = cv2.VideoWriter_fourcc(*'XVID')

            video_file = os.path.join(
                OUTPUT_PATH, f'{ALPHA}_{TM}.avi')
            video_writer = cv2.VideoWriter(
                video_file, fourcc, 10, (im.shape[1]*3, im.shape[0]))

            bg_scores = []
            fg_scores = []
            # Main loop
            for fr in range(n):
                # Read the image
                im = cv2.imread(os.path.join(INPUT_PATH, flist[fr]))

                # Classify the foreground using the model & Update the model with the new image
                fg_mask = bg_model.classify(im)
                bg_model.update(im)

                # Display the input frame, background model, and foreground mask
                im_draw = im.copy()
                cv2.putText(im_draw, f'Frame {fr+1}/{n}', (im.shape[1]//15, 20),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
                cv2.putText(im_draw, f'ALPHA = {ALPHA}, TM = {TM}', (im.shape[1]//15, 40),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)

                bg_draw = bg_model.mean.astype('uint8')
                cv2.putText(bg_draw, f'Frame {fr+1}/{n}', (im.shape[1]//15, 20),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
                cv2.putText(bg_draw, f'ALPHA = {ALPHA}, TM = {TM}', (im.shape[1]//15, 40),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)

                fg_draw = cv2.cvtColor(fg_mask, cv2.COLOR_BGR2GRAY)
                cv2.putText(fg_draw, f'Frame {fr+1}/{n}', (im.shape[1]//15, 20),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
                cv2.putText(fg_draw, f'ALPHA = {ALPHA}, TM = {TM}', (im.shape[1]//15, 40),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)

                cv2.imshow('Input Frame', im_draw)
                cv2.imshow('Background Model', bg_draw)
                cv2.imshow('Foreground Mask', fg_draw)

                # Press 'q' to exit the loop
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break

                # Save the results for specific frames
                if fr in [5, 100, 400]:
                    fname = f'FGmask_{ALPHA}_{TM}_{fr}.png'
                    fname_wpath = os.path.join(OUTPUT_PATH, fname)
                    cv2.imwrite(fname_wpath, fg_draw)

                    fname = f'BGmean_{ALPHA}_{TM}_{fr}.png'
                    fname_wpath = os.path.join(OUTPUT_PATH, fname)
                    cv2.imwrite(fname_wpath, bg_draw)

                # Write the current frame and FGmask to the videos
                frame = np.hstack(
                    (im_draw, bg_draw, cv2.cvtColor(fg_draw, cv2.COLOR_GRAY2BGR)))
                video_writer.write(frame)

                # Print the progress using the same line
                score_bg, score_fg = bg_model.evaluate()
                bg_scores.append(np.mean(score_bg))
                fg_scores.append(np.mean(score_fg))

                print(f'Frame: {fr}/{n}, bg%: {np.mean(score_bg):.5f}, fg%: {np.mean(score_fg):.5f}', end='\r')

            # conbine these eix images into one image
            frame_numbers = [5, 100, 400]
            file_prefixes = ['FGmask', 'BGmean']
            image = combine_images(
                OUTPUT_PATH, ALPHA, TM, frame_numbers, file_prefixes)

            fname = f'{ALPHA}_{TM}.jpg'
            fname_wpath = os.path.join(OUTPUT_PATH, fname)
            cv2.imwrite(fname_wpath, image)

            cv2.destroyAllWindows()
            video_writer.release()
            cv2.destroyAllWindows()

            # print the status
            print(
                f'Done, bg%: {np.mean(score_bg):.5f}, fg%: {np.mean(score_fg):.5f}   \n')

            plt.plot(range(0, fr + 1), bg_scores, label='Background Score')
            plt.plot(range(0, fr + 1), fg_scores, label='Foreground Score')
            plt.xlabel('Frame Number')
            plt.ylabel('Score')
            plt.legend()
            plt.savefig(os.path.join(
                OUTPUT_PATH, f'op_curve_{ALPHA}_{TM}.png'))
            # then end this plt
            plt.close()


if __name__ == '__main__':
    main()
```

Running dialog:

ALPHA = 0.01, TM = 2
Done, bg%: 0.94395, fg%: 0.84446   0.84446

ALPHA = 0.01, TM = 3
Done, bg%: 0.94395, fg%: 0.92052   0.92052

ALPHA = 0.01, TM = 4
Done, bg%: 0.94395, fg%: 0.96885   0.96885

ALPHA = 0.01, TM = 5
Done, bg%: 0.94395, fg%: 0.99053   0.99053

ALPHA = 0.01, TM = 6
Done, bg%: 0.94395, fg%: 0.99669   0.99669

ALPHA = 0.003, TM = 2
Done, bg%: 0.95890, fg%: 0.97283   0.97283

ALPHA = 0.003, TM = 3
Done, bg%: 0.95890, fg%: 0.99760   0.99760

ALPHA = 0.003, TM = 4
Done, bg%: 0.95890, fg%: nan   g%: nan

ALPHA = 0.003, TM = 5
Done, bg%: 0.95890, fg%: nan   g%: nan9034

ALPHA = 0.003, TM = 6
Done, bg%: 0.95890, fg%: nan   g%: nan9209

ALPHA = 0.001, TM = 2
Done, bg%: 0.96782, fg%: 0.98956   0.98956

ALPHA = 0.001, TM = 3
Done, bg%: 0.96782, fg%: 0.99823   0.99823

ALPHA = 0.001, TM = 4
Frame: 578/611, bg%: 0.96783, fg%: 0.97714

ALPHA = 0.01, TM = 2
Frame: 453/611, bg%: 0.92858, fg%: 0.81196

ALPHA = 0.0003, TM = 3
Done, bg%: 0.96294, fg%: 0.99392   0.99392

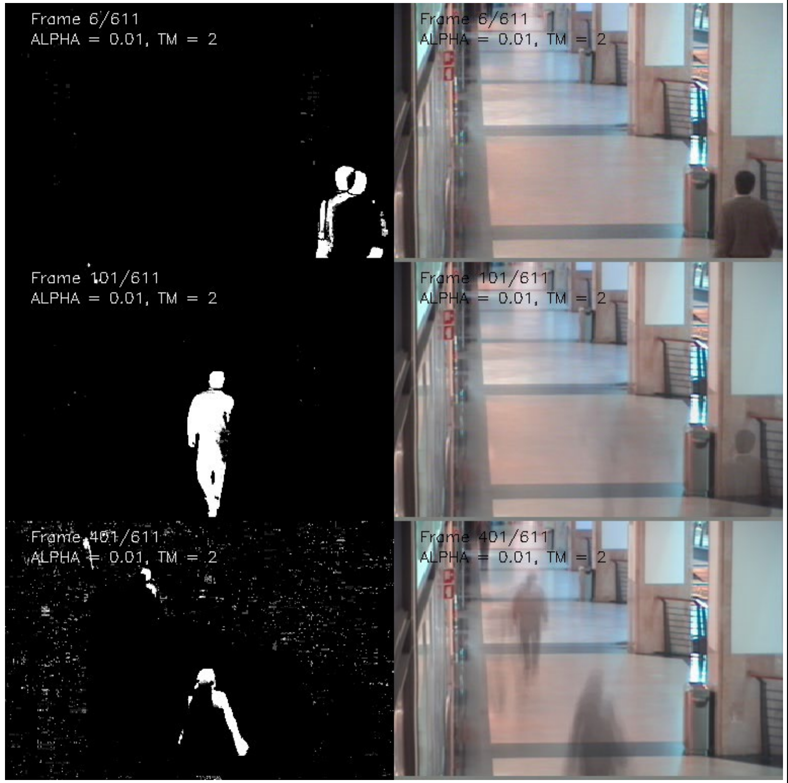
ALPHA = 0.0003, TM = 4
Done, bg%: 0.96294, fg%: nan   g%: nan8552

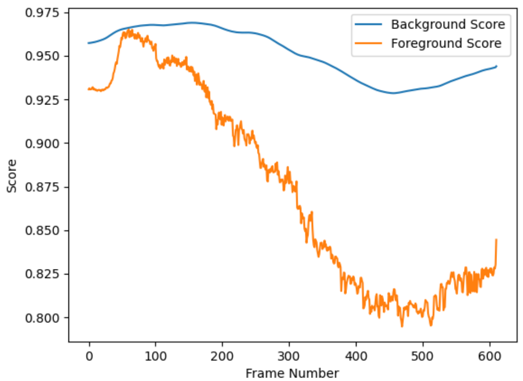
ALPHA = 0.0001, TM = 2
Done, bg%: 0.95944, fg%: 0.93437   0.93437

ALPHA = 0.0001, TM = 3
Done, bg%: 0.95944, fg%: 0.95644   0.95644

ALPHA = 0.0001, TM = 4
Done, bg%: 0.95944, fg%: 0.99379   0.99379

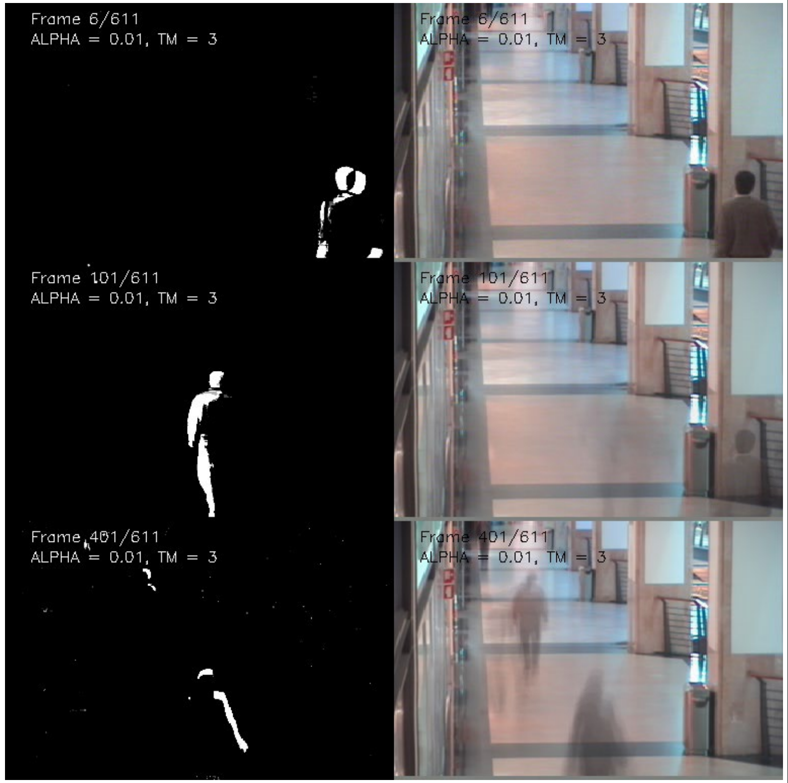
Results 1. Alpha = 0.01

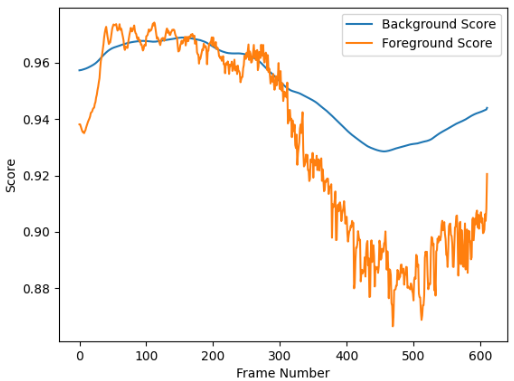




Alpha = 0.01

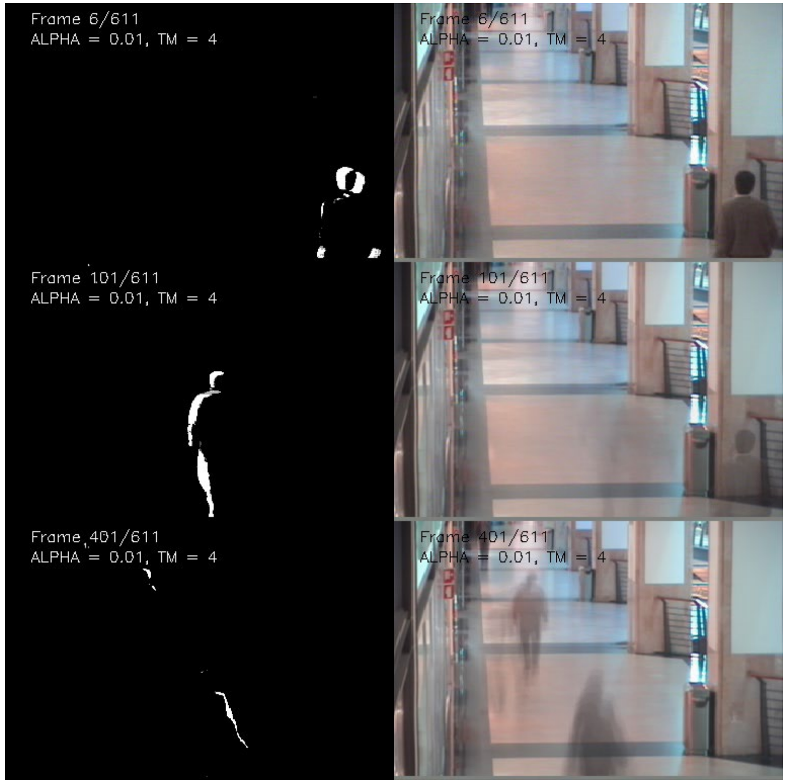
TM = 2

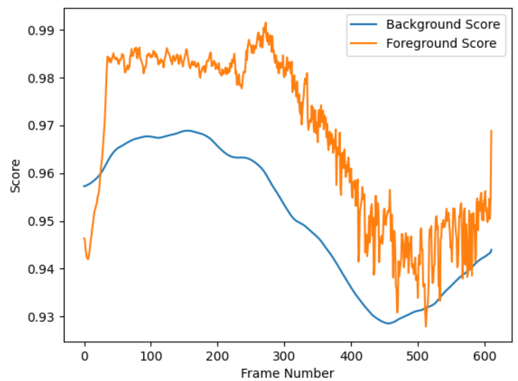




Alpha = 0.01

TM = 3

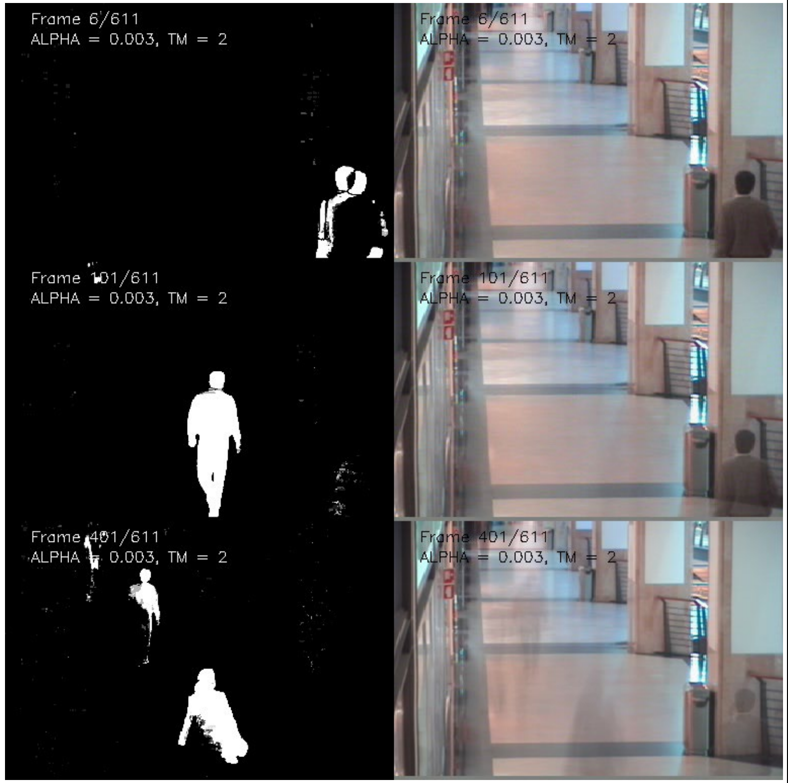


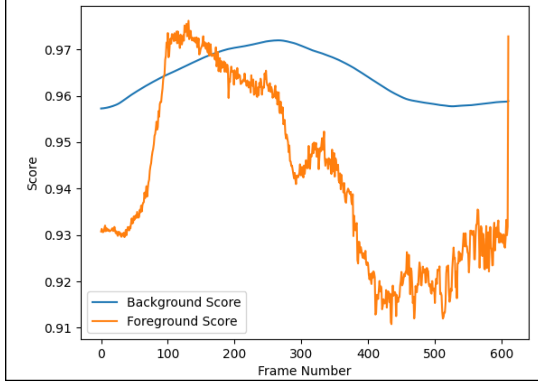


Alpha = 0.01

TM = 4

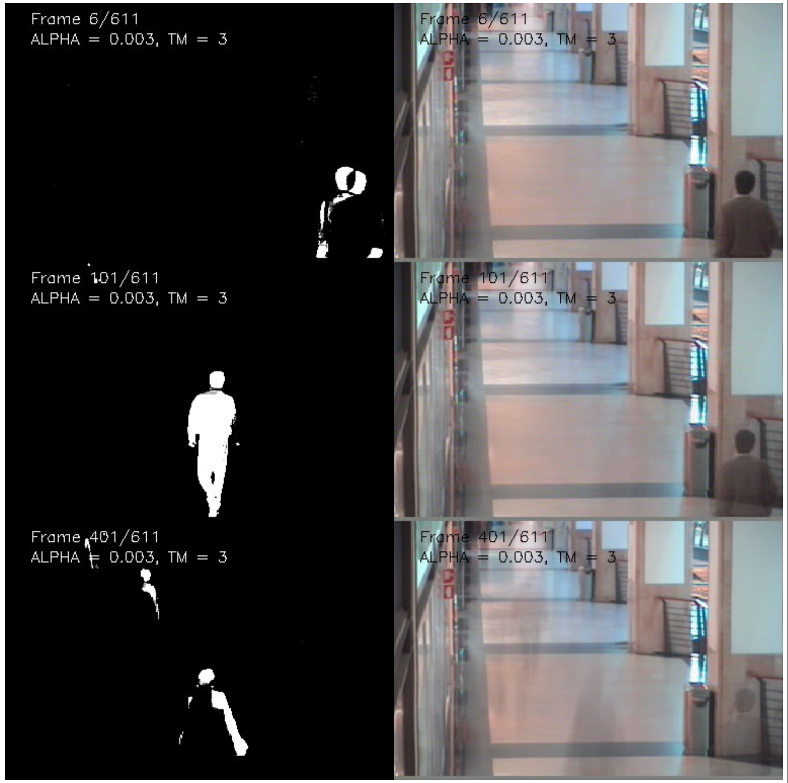
Results 2. Alpha = 0.003

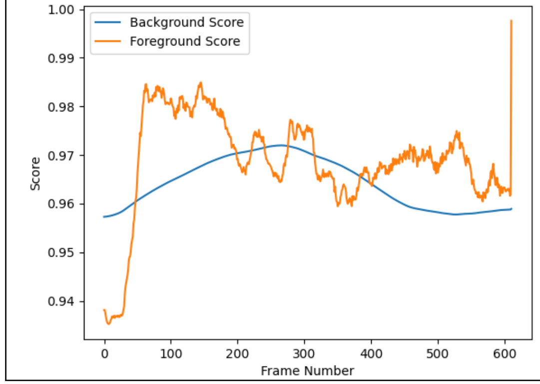




Alpha = 0.003

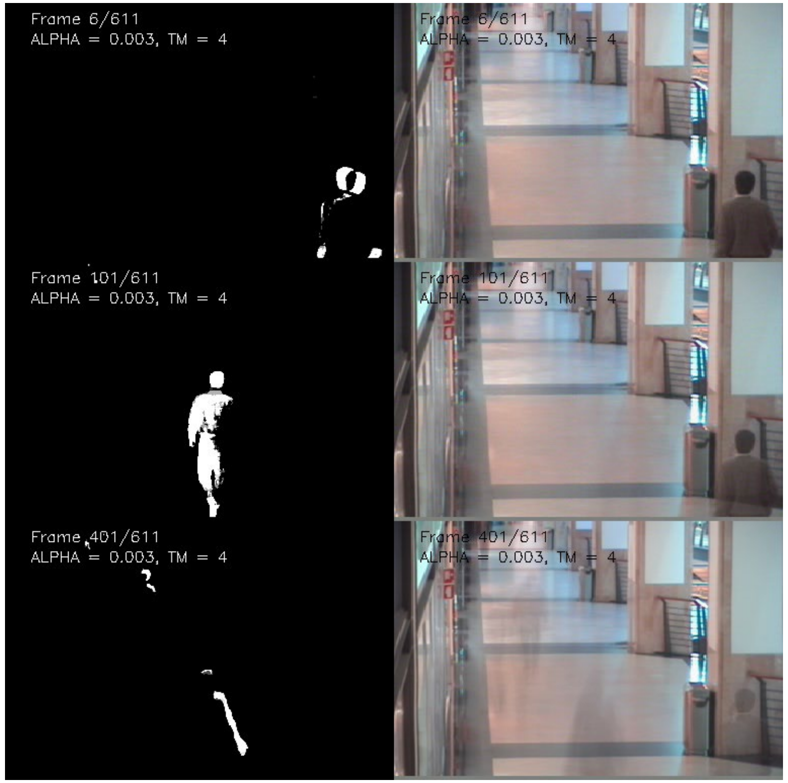
TM = 2

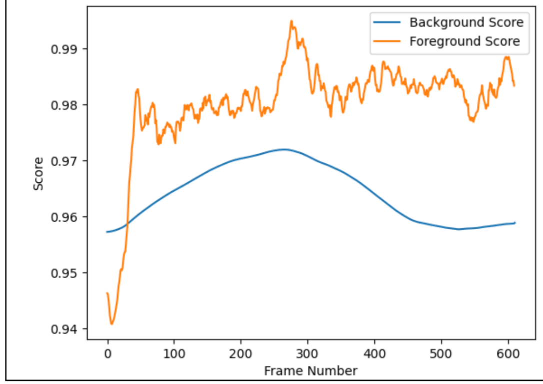




Alpha = 0.003

TM = 3

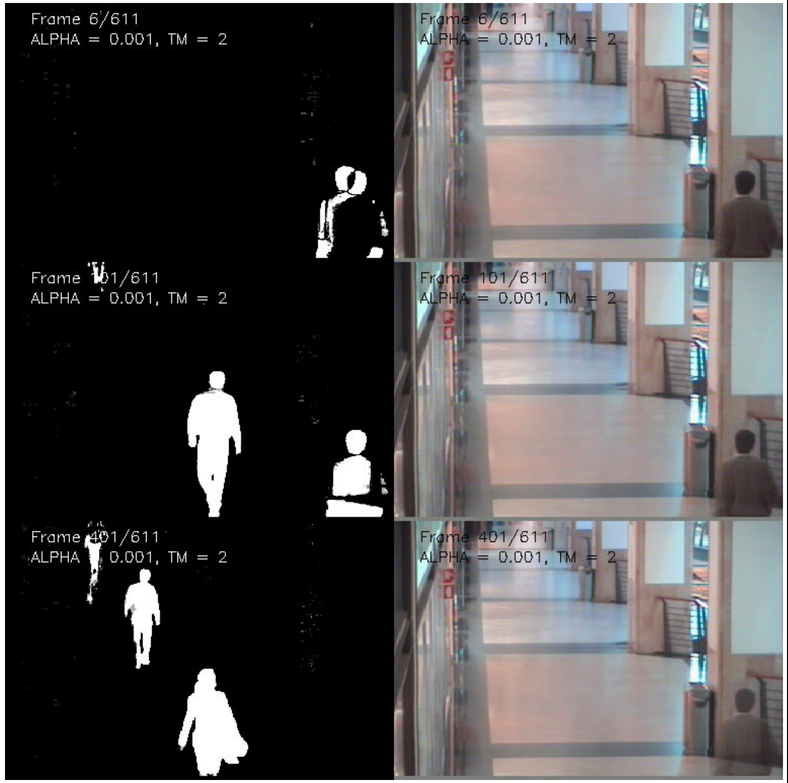


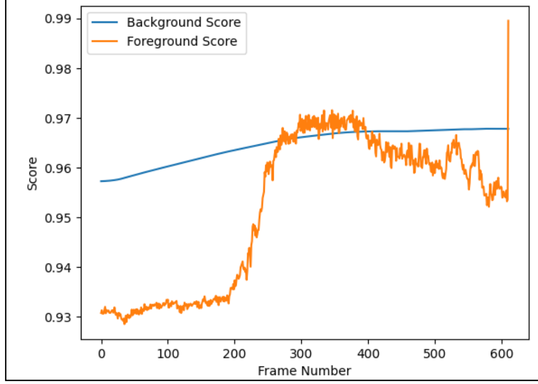


Alpha = 0.003

TM = 4

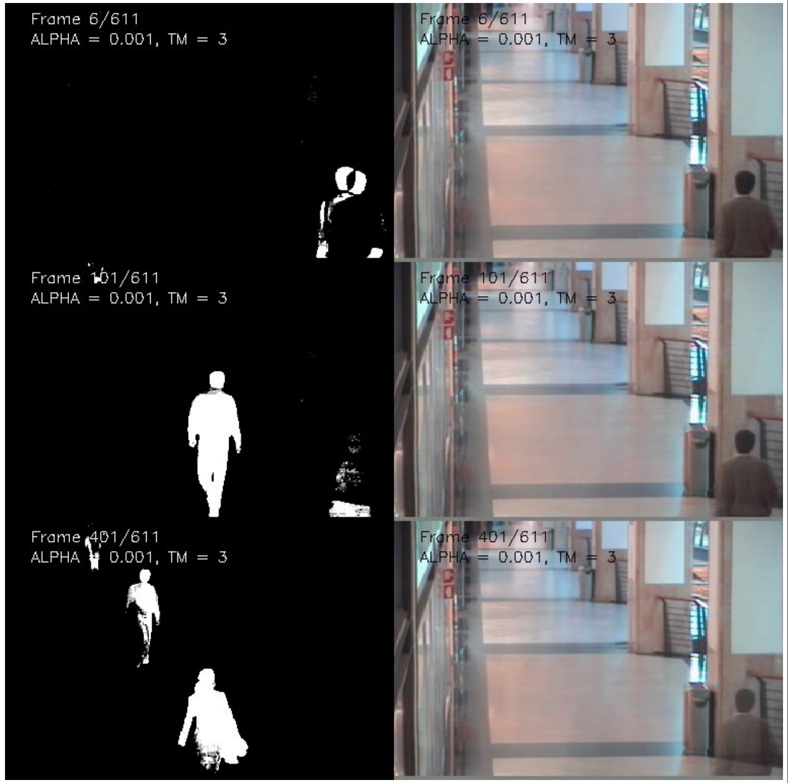
Results 3. Alpha = 0.001

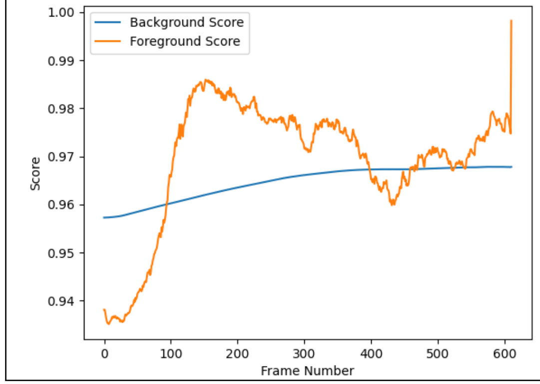




Alpha = 0.001

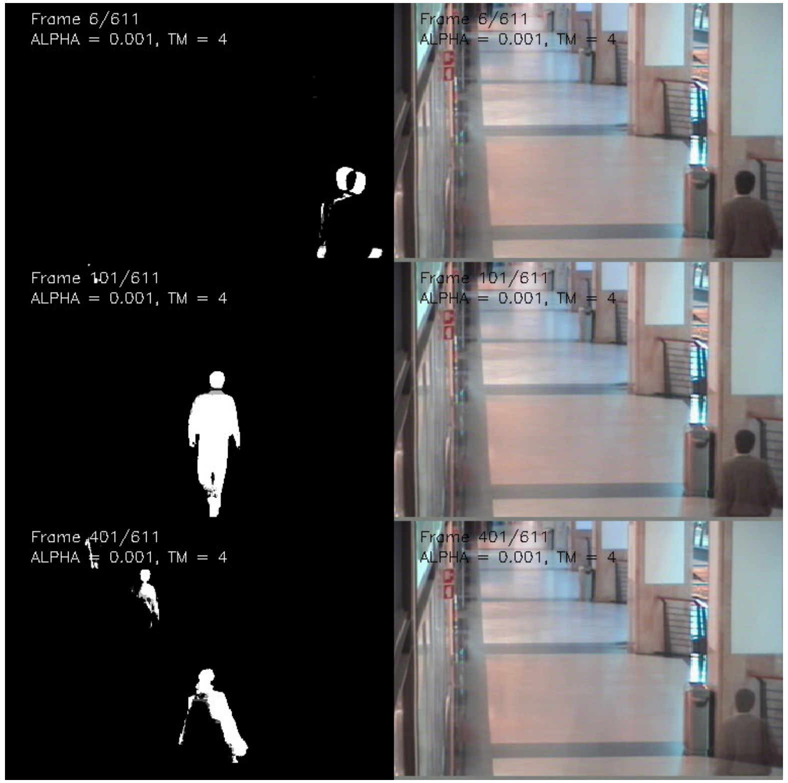
TM = 2

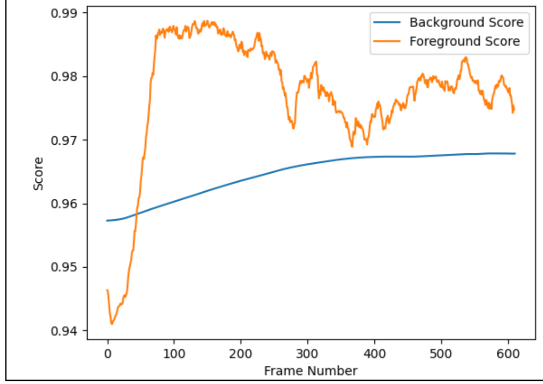




Alpha = 0.001

TM = 3

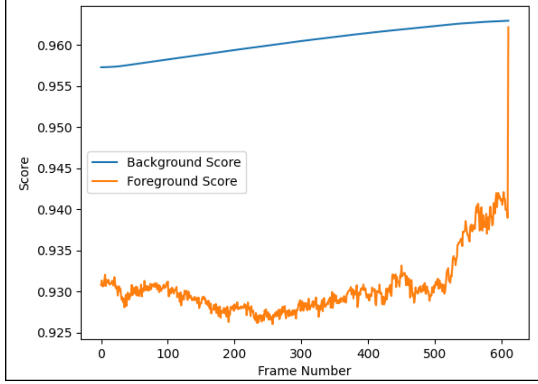




Alpha = 0.001

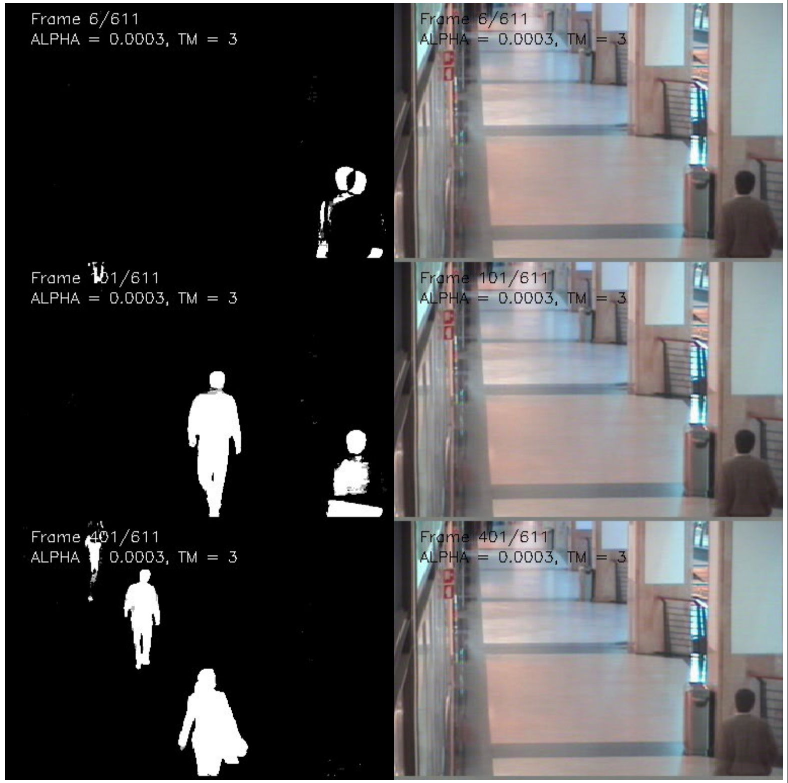
TM = 4

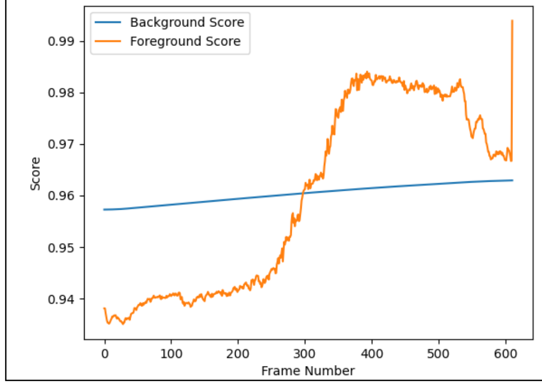
Results 4. Alpha = 0.0003



Alpha = 0.0003

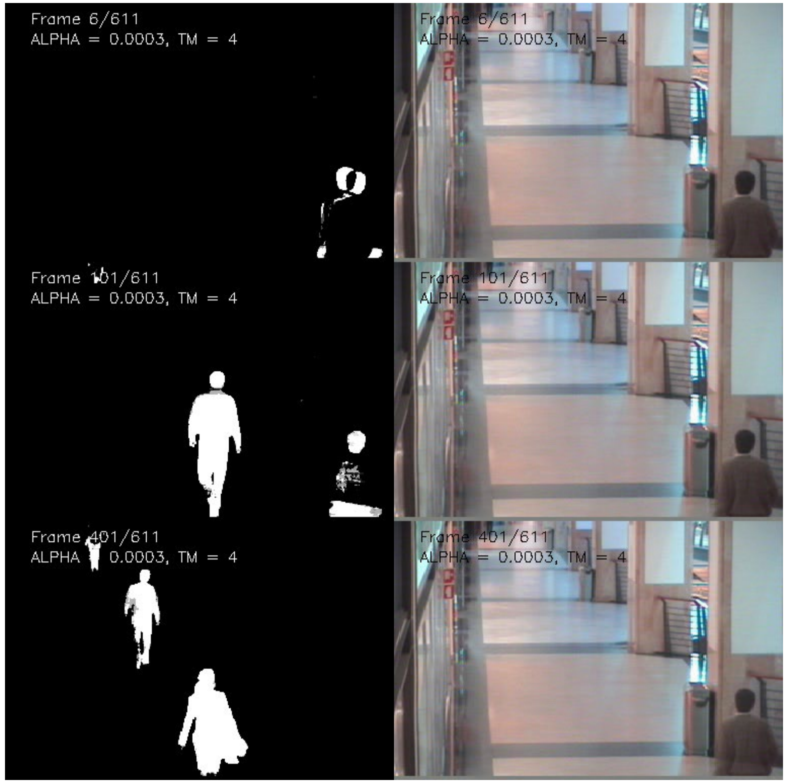
TM = 2

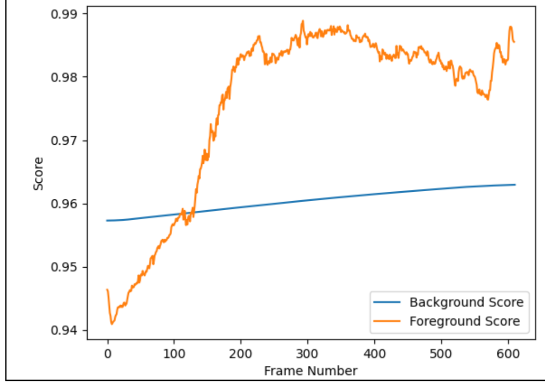




Alpha = 0.0003

TM = 3

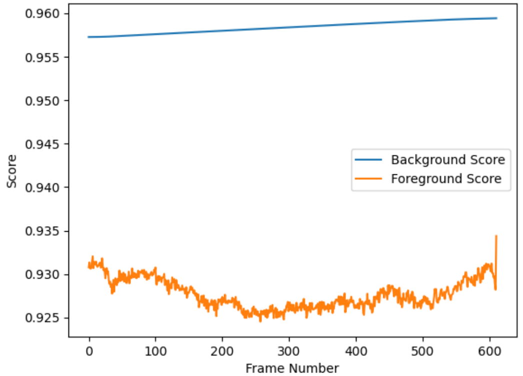




Alpha = 0.0003

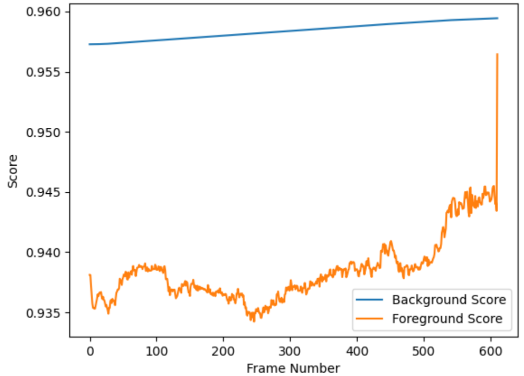
TM = 4

Results 5. Alpha = 0.0001



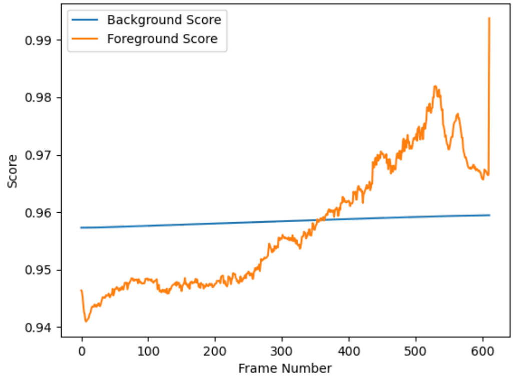
Alpha = 0.0001

TM = 2



Alpha = 0.0001

TM = 3



Alpha = 0.0001

TM = 4

Results Interpretation & Discussion

In this implementation of a simple background subtraction algorithm for detecting moving objects using a stationary camera, different learning rates (alpha) and matching thresholds (tm) are tested to observe their impact on the algorithm's performance. The chosen values are:

ALPHA_list = [0.01, 0.003, 0.001, 0.0003, 0.0001]

TM_list = [2, 3, 4]

Learning Rate (*alpha*):

The learning rate (**alpha**) determines how quickly the background model adapts to changes in the scene. A higher alpha value means the model will update faster and be more sensitive to rapid changes in the scene, whereas a lower alpha value will cause the model to update slowly, being less sensitive to scene changes.

From the experiments conducted with various alpha values, it can be observed that:

A high **alpha** (e.g., 0.01) results in a more adaptive model that quickly reacts to changes in the scene. This can lead to better detection of moving objects in dynamic environments. However, this can also result in false detections due to noise or small variations in lighting and other scene factors.

A low **alpha** (e.g., 0.0001) leads to a more stable background model that is less prone to false detections due to noise. However, the model will be slower to adapt to actual changes in the scene, possibly resulting in poor object detection when the scene is dynamic or when objects are moving slowly.

Matching Threshold (*TM*):

The matching threshold (**TM**) is used to determine whether a pixel belongs to the background or foreground based on the difference between its current value and the background model. A higher threshold means that a pixel must have a larger difference from the background model to be considered foreground, and a lower threshold makes the classification more sensitive to smaller differences.

From the experiments conducted with various **TM** values, it can be observed that:

A low **TM** (e.g., 2) results in more sensitivity to differences between the current frame and the background model, potentially detecting more foreground objects. However, this increased sensitivity can lead to more false detections due to noise or small variations in the scene.

A high **TM** (e.g., 4) reduces the sensitivity to differences between the current frame and the background model, potentially reducing false detections. However, this can also result in missed detections of actual moving objects if their differences from the background model are subtle.

In conclusion, the choice of learning rate (**alpha**) and matching threshold (**TM**) significantly impacts the performance of the background subtraction algorithm. The optimal values for these parameters depend on the specific application and the nature of the scene being analyzed. Tuning these parameters for the specific use case can improve the accuracy and robustness of the algorithm in detecting moving objects.