Motion Detection Using Simple Image Filtering

By- Arvinder Singh and Uday Raghuvanshi

ABSTRACT

The aim of the project is to explore a technique for motion detection in image sequences captured with a stationary camera. Here, most of the pixels belong to a stationary background and relatively small moving objects pass in front of the camera. The intensity values at a pixel are either constant or slowly varying except when a moving object begins to pass through that pixel, replacing background intensity by foreground intensity. Thus, we aim to look at large gradients in the temporal evolution of pixel values.

We started by reading in the images and making them grayscale, followed by computing the temporal derivative of each consecutive frame. Then we used a threshold value to create a 0 and 1 mask of moving objects. Finally, we combined the mask with the original frame to display the results. In order to add variations to our above algorithm, we implemented 3 temporal derivative filters: A) prewitt [-1,0,1], B) simple 0.5*[-1,0,1] and C) 1D derivative of a Gaussian with varying sigma and compared the results. We also performed 2D spatial smoothing filtering to the frames in order to reduce noise, before applying a temporal derivative. We implemented 3 smoothing filters: A) 3x3 Box filter, B) 5x5 Box filter and C) 2D Gaussian filter with varying sigma, and compared the results we got. Next, we start changing the threshold value to see which result works best for us. Finally, we try to implement a strategy that helps us select a good threshold for each image.

Description of algorithm

In order to implement motion detection, we started by forming a MotionDetection class which has several methods which perform the different steps involved in motion detection. We have used python as the programming language and used modules including cv2, numpy and os. The instance of the class takes ‘images_path’ and ‘type_derivative mask’ as inputs.

The description of each method of our algorithm is as follows:

1. **img_array:** This method reads images from the folder, converts them into gray scale and stores in an array called ‘img_arr’

```
def img_array(self):
    imgs_arr = []
    for img_name in os.listdir(self.images_path):
        img = cv2.imread(self.images_path + '\\' + img_name, 0)
        imgs_arr.append(np.asarray(img).astype(int))
    return imgs_arr
```

2. **temporal_derivative:** This method takes the img_arr as input, loops over the entire array and performs a temporal derivative of consecutive frames. The type of derivative depends on the user input ‘prewitt’, ‘simple’ or ‘gauss’. It outputs an array containing the temporal derivative of images.

```
def temporal_derivative(self, imgs_arr: list):
    i = 0
    temporal_derivative = []

    while i < len(imgs_arr):
        if 1 < i:
            temp_img_0 = imgs_arr[i - 2]
            temp_img_2 = imgs_arr[i]
            if self.type_derivative_mask == "prewitt":
                temporal_derivative.append(np.subtract(temp_img_2, temp_img_0))
            elif self.type_derivative_mask == "simple":
                temporal_derivative.append(0.5*np.subtract(temp_img_2, temp_img_0))
            else:
                sigma = 0.3
                x_0 = (-(-1) / (np.square(sigma))) * np.exp(-(np.square(-1)) / (2 * np.square(sigma)))
                x_2 = (-1 / (np.square(sigma))) * np.exp(-(np.square(1)) / (2 * np.square(sigma)))
                temporal_derivative.append(np.add(x_2 * temp_img_2, x_0 * temp_img_0))
        i += 1

    return temporal_derivative
```

3. **filtering:** This static method takes as input the filter_type and the img_array. The filter_type depends on the user input ‘3x3’, ‘5x5’ or ‘gauss’. It returns an array of filtered images (filtered_imgs).

```
@staticmethod
def filtering(filter_type: str, imgs_arr: list):
    filtered_imgs = []
    if filter_type == '3x3':
        kernel1 = np.ones((3, 3), np.float32)/9
        for i in imgs_arr:
            filtered_imgs.append(cv2.filter2D(i.astype(np.uint8), ddepth=-1, kernel=kernel1).astype(int))
    elif filter_type == '5x5':
        kernel2 = np.ones((5, 5), np.float32)/25
        for i in imgs_arr:
            filtered_imgs.append(cv2.filter2D(i.astype(np.uint8), ddepth=-1, kernel=kernel2).astype(int))
    else:
        for i in imgs_arr:
            filtered_imgs.append((cv2.GaussianBlur(i.astype(np.uint8), (7, 7), 0.5)).astype(int))
    return filtered_imgs
```

4. thresholding: This static method takes the temporal_derivative array and thresholding type (type_of_thresh) as input. If we input the type_of_thresh as 'max', then it returns the maximum absolute value of the temporal gradient of the stationary pictures which we can use as a threshold. If we input the type_of_thresh as 'std' then the method returns the standard deviation of the temporal gradient of the stationary pictures which we can use as the threshold and an input to the combine mask method.

```
@staticmethod
def thresholding(temporal_derivative: list, type_of_thresh: str):

    if type_of_thresh == 'max':
        mx = 0
        mn = 0
        for td in temporal_derivative:
            if np.max(td) > mx:
                mx = np.max(td)
            if np.min(td) < mn:
                mn = np.min(td)
            return max(abs(mx), abs(mn))

    elif type_of_thresh == 'std':
        ideal_thresh = np.zeros(np.shape(temporal_derivative[0]))
        var_arr = np.zeros(np.shape(temporal_derivative[0]))

        for img in temporal_derivative:
            var_arr = var_arr + np.square(ideal_thresh - img)
        std_dev_arr = np.sqrt(var_arr * 1 / (len(temporal_derivative) - 1))
        average_std_dev = np.mean(std_dev_arr)
        return average_std_dev
```

5. combine_mask: This static method forms a 0 and 1 mask of moving objects based on the input threshold value and combines the mask with the original frame. It takes the temporal_derivative, imgs_arr and the value of threshold as input. It returns an array of masked images as output.

```
@staticmethod
def combine_mask(temporal_derivative: list, imgs_arr: list, threshold: float):
    masked_imgs = []
    imgs_arr.pop(0)
    imgs_arr.pop(-1)
    i = 0
    for temp_der_i in temporal_derivative:
        temporal_derivative = np.where(abs(temp_der_i) > threshold, 1, 0)
        masked_imgs.append(np.multiply(imgs_arr[i].astype(np.uint8), temporal_derivative.astype(np.uint8)))
        i += 1
    return masked_imgs
```

Experiments:

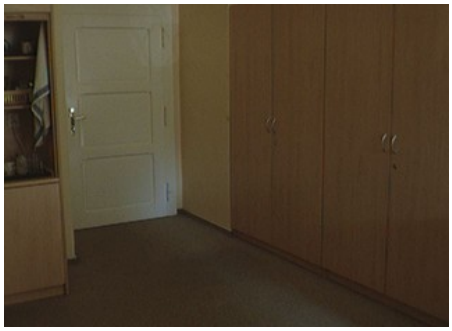
Without Filtering images

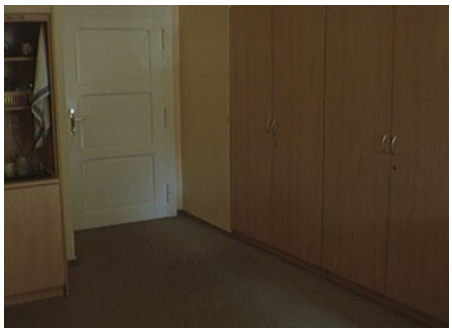
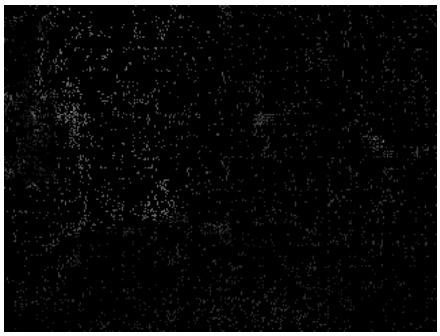
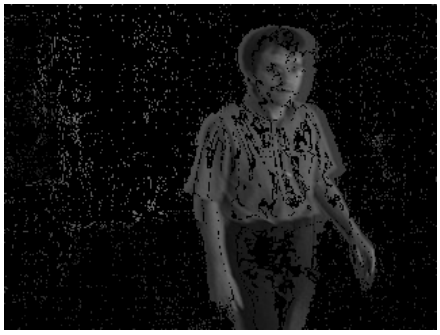
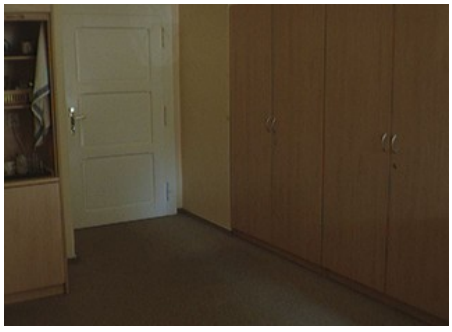
- **Temporal Derivative using different filters:-** Here we are applying different derivative filters to compute the temporal gradient and then thresholding at different values

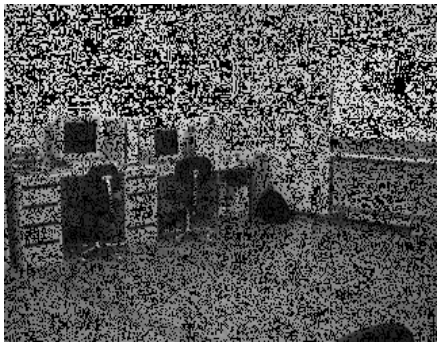
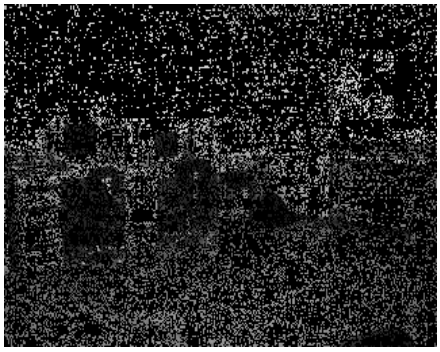
1. **1D prewitt mask**

-1	0	1

Comparing two different data sets for different threshold values-

Threshold	Original Image	Masked Image
1		
		

3.5		
		
12		
		

Threshold	Original Image	Masked Image
1		
		
3.5		
		

12		
		

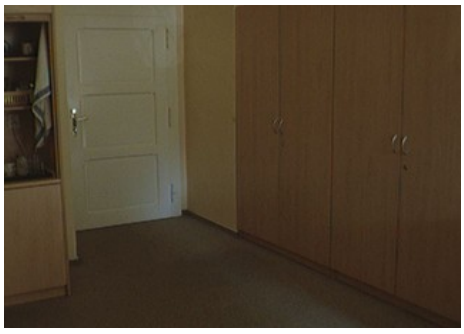
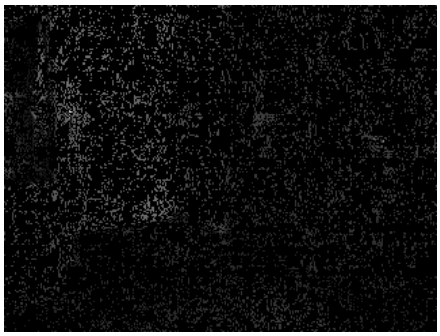
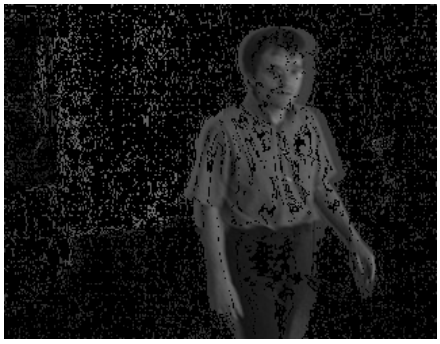
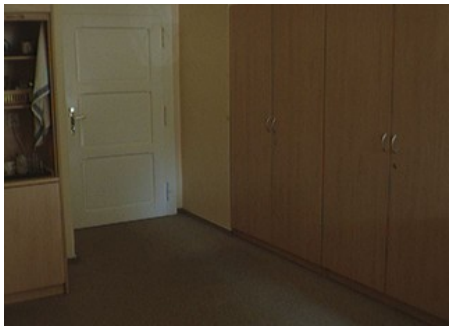
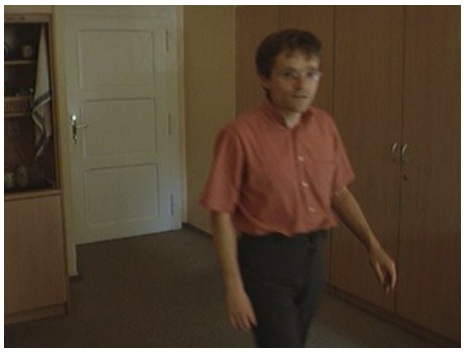
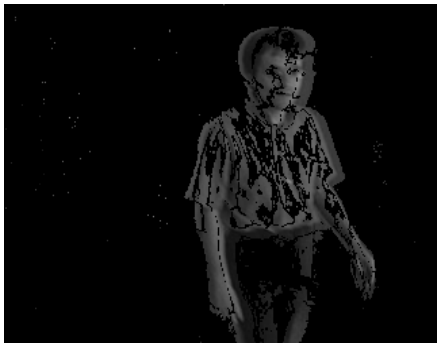
Fig: Output images upon varying thresholds for prewitt mask

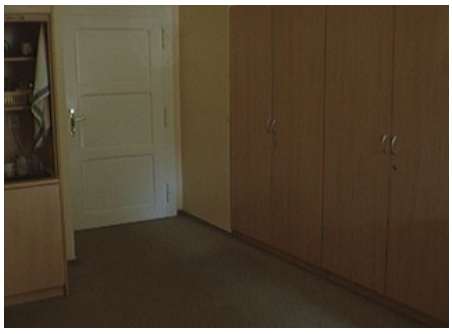
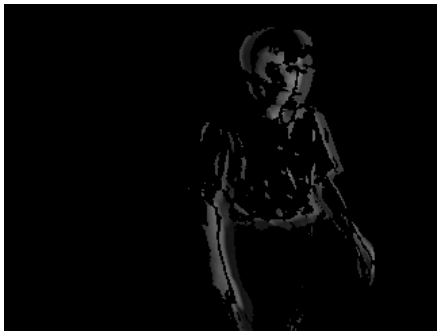
2. Simple 1D mask

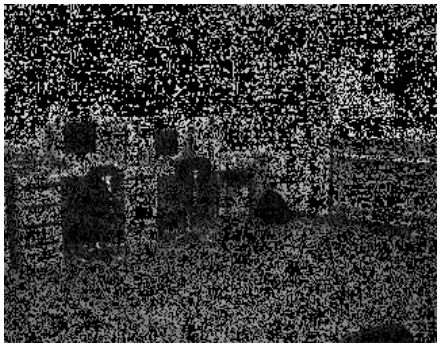
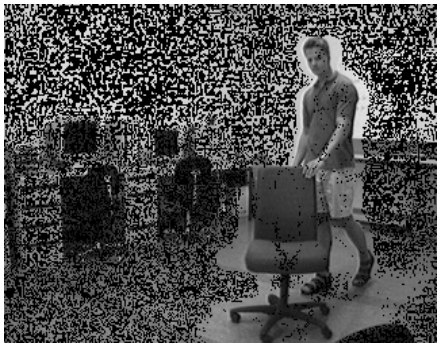
-0.5	0	0.5

Comparing two different data sets for different threshold values-

Threshold	Original Image	Masked Image

1	 	 
3.5	 	 

12	 	 

Threshold	Original Image	Masked Image
1	 	 

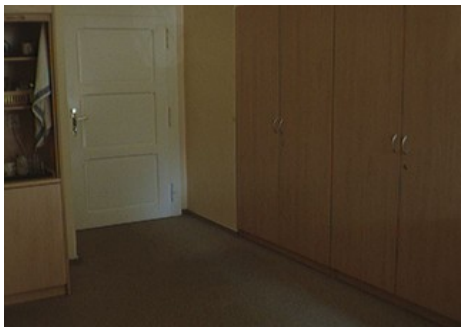
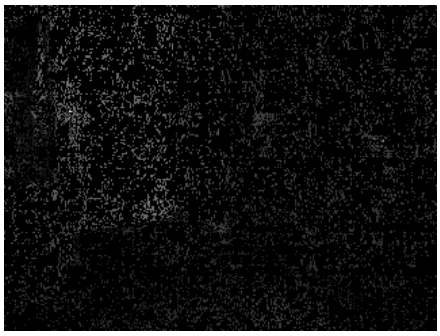
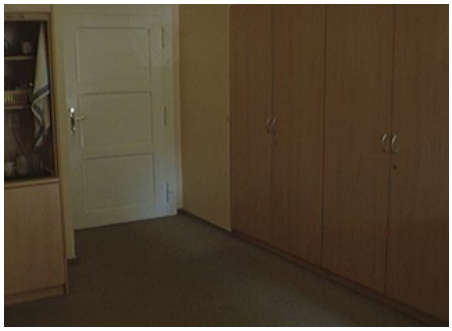
3.5		
		
12		
		

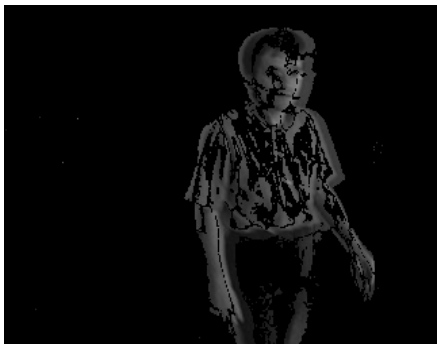
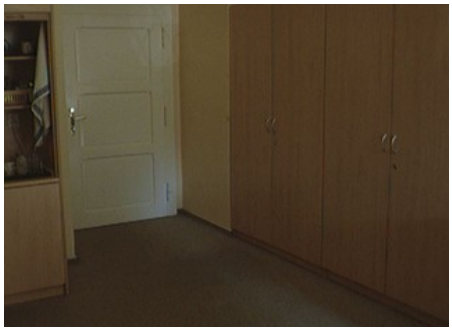
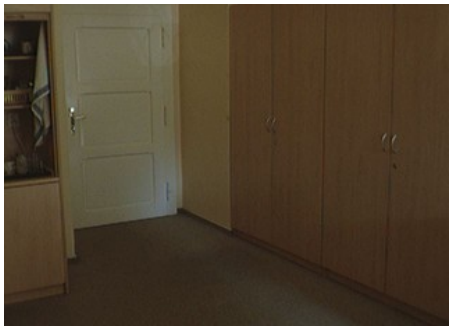
Fig: Output images upon varying thresholds for simple mask

3. Derivative of a gaussian mask

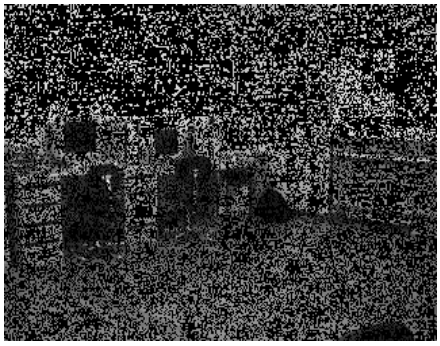
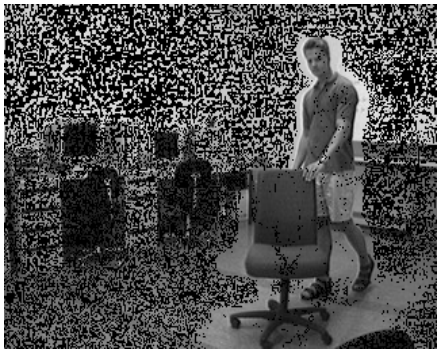
$\frac{e^{\frac{-1}{2\sigma^2}}}{2\sigma^2}$	0	$\frac{\frac{-1}{2\sigma^2}}{\frac{-e}{2\sigma^2}}$

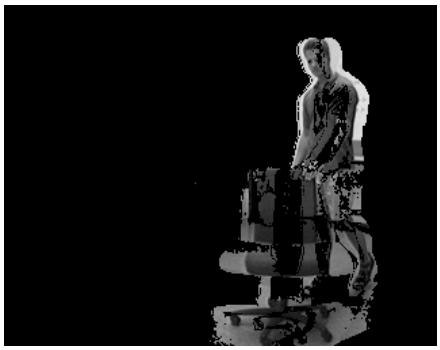
Comparing two different data sets for different threshold values (tsigma = 0.3)

Threshold	Original Image	Masked Image
0.1		
		
0.4		

		
1		
		
3.5		

Threshold	Original Image	Masked Image
0.1		
		

0.4		
		
1		
		

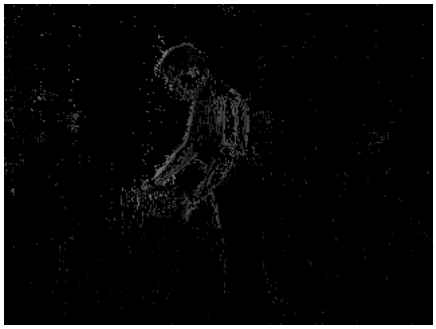
3.5	 	 

Fig: Output images upon varying thresholds for gaussian mask

Comparing results of 1D derivative of a gaussian filter using different values of std. deviation($\boldsymbol{\sigma}$). We are assuming a constant **threshold of 0.2.**

Dataset 1

Masked Image ($\boldsymbol{\sigma}$ = 0.1)	Masked Image ($\boldsymbol{\sigma}$ = 1)	Masked Image ($\boldsymbol{\sigma}$ = 10)
		

Masked Image (σ = 0.5)	Masked Image (σ = 5)	Masked Image (σ = 15)
		

Dataset 2

Masked Image (σ = 0.1)	Masked Image (σ = 1)	Masked Image (σ = 10)
		

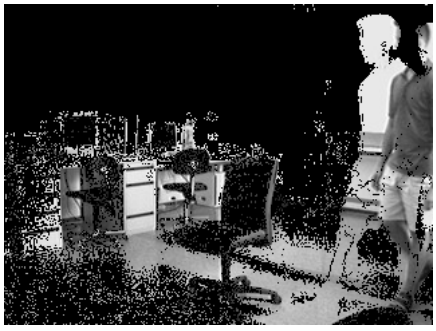
Masked Image (σ = 0.5)	Masked Image (σ = 5)	Masked Image (σ = 15)
		

Fig: Output images upon varying thresholds for varying σ of gaussian mask

Observations

- **Comparing Prewitt - Simple - Derivative of a Gaussian filters**

We start by calculating the Standard deviation of the temporal gradient of the first 'n' images that are stationary(have no motion of objects). In the case of the Office dataset, we are using the first 40 images and for the RedChair dataset, we are using the first 20 images to calculate the std deviation of the temporal gradient.

Office Dataset

Std Deviation (Prewitt) = 1.8584274813420358

Std Deviation (Simple) = 0.9292137406710179

Std Deviation (Gaussian $\sigma_{0.3}$) = 0.0798281358652211

RedChair Dataset

Std Deviation (Prewitt) = 3.891491703692509

Std Deviation (Simple) = 1.9457458518462545

Std Deviation (Gaussian $\sigma_{0.3}$) = 0.16715773499884687

Using a 1D derivative of a gaussian mask to compute the temporal derivative reduces the most amount of noise, then the simple filter. There is no noise reduction using a prewitt filter and we can see in our experimental results that the threshold for temporal gradient using a Prewitt filter which gives us good results is the highest. This is because the variance of noise is larger using a prewitt filter because of no noise reduction.

Every derivative filter has its own threshold value for which it gives good results, the higher threshold we use the more false negatives we'll get and similarly the lower threshold we use the more false positives we'll get. We can see in the experimental results, if we use a 3.5 threshold for a 1D derivative of a gaussian filter, we'll get a lot of false negatives.

One thing to note here is that, we have to choose the sigma of the 1D gaussian derivative filter carefully. If we reduce the noise a lot, then it will be very sensitive to changes in the gradient. For a prewitt or a simple mask, the tolerance is a bit more than the gaussian mask.

- **Comparing different values of standard deviation for derivative of a Gaussian mask**

We observed that using a very small or a very large standard deviation results in a large reduction in noise and a very small threshold. The problem with this is that it is very sensitive to a small change in the value of temporal gradient. We can see in the results that for a very small or a very large value of standard deviation, we observe more false negatives and the output is a black image with rarely any 1 values(for the same threshold).

As we increase the standard deviation, the noise is lowered up to a certain value after that the noise starts reducing again which also makes sense as we have seen in the class- the graph of derivative of a gaussian increases-decreases and then increases again.

Let's take an example of the Office dataset for the first 40 images(stationary).

Std Deviation (Gaussian $\sigma_{0.1}$) = 3.564570731411361e-20

Std Deviation (Gaussian $\sigma_{0.6}$) = 1.2800945564682378

Std Deviation (Gaussian σ_{1}) = 1.1209444498972208

Std Deviation (Gaussian σ_{5}) = 0.07246118527036371

Std Deviation (Gaussian σ_{15}) = 0.008195655881605642

After filtering Images

1. **Using Prewitt derivative:** Here we keep the value of threshold as 2

Filter type	Before filtering	After filtering	Masked image
3x3 Box filter			
5x5 Box filter			
2D gaussian filter with sigma=0.1			
2D gaussian filter with sigma=1			

2D gaussian filter with sigma=10			

Fig: Table showing the difference of results using different amounts of smoothing (for prewitt derivative)

2. **<u>Using Simple derivative:</u>** Here we keep the constant value of threshold as 1.5

Filter type	**Before filtering**	**After filtering**	**Masked image**
3x3 Box filter			
5x5 Box filter			
2D gaussian filter with sigma=0.1			

2D gaussian filter with sigma=1			
2D gaussian filter with sigma=10			

Fig: Table showing the difference of results using different amounts of smoothing (for simple derivative)

3. **<u>Using gaussian derivative:</u>** Here we keep the constant value of threshold as 0.1. Also, the value of sigma for gaussian derivatives is 0.3 (same as part (i)).

Filter type	**Before filtering**	**After filtering**	**Masked image**
3x3 Box filter			
5x5 Box filter			

2D gaussian filter with sigma=0.1			
2D gaussian filter with sigma=1			
2D gaussian filter with sigma=10			

Fig: Table showing difference of results using different amounts of smoothing (for gaussian derivative)

Observations

- **Comparing Box filters and Gaussian filters for Prewitt derivative**

We start by filtering the images with one filter at a time and calculating the standard deviation of temporal gradient of first 'n' images for each case (n=20 for 'Office' and n=40). Calculating standard deviation helped us to estimate which case produced better results by measuring the noise. We know that the first few are of a stationary scene and should ideally have the same pixel intensities. However, the noise in frames makes the reality different from what we expect. We calculated the values and performed experiments on the 'Office' dataset to keep results concise and readable.

Office Dataset

Std Deviation (3x3) = 0.9339165115322813

Std Deviation (5x5) =0.6557720547813229

Std Deviation (Gaussian $\boldsymbol{\sigma}_{0.1}$) =1.8474352154931075

Std Deviation (Gaussian $\boldsymbol{\sigma}_{1}$) = 0.858430875010392

Std Deviation (Gaussian $\boldsymbol{\sigma}_{10}$)= 0.6556804940321004

We learnt in class that smoothing filters help in reducing noise by averaging the values at each pixel with respect to neighbours (in case of box filters) and by weighing each pixel by a gaussian distribution (in case of gaussian filters). Upon applying a 3x3 box filter, we observe that the image is less blurred and hence less noise is removed when compared to a 5x5 box filter. This statement is supported by the fact that we get more std deviation for the first 20 frames of filtered images in case of 3x3 filter. As we increase the size of the filter, we increase the window of averaging and thus pixels in the background where pixel values don't vary much tend to be more zeroed out in the case of 5x5. The table above shows the difference in images obtained.

When we use gaussian filters with varying sigma, we observe that the filter with less value of $\boldsymbol{\sigma}$ (0.1) reduces less noise than the filter with high $\boldsymbol{\sigma}$ (10). The filtered images show that higher $\boldsymbol{\sigma}$ blurries the image more than lower $\boldsymbol{\sigma}$. Thus, filter with higher $\boldsymbol{\sigma}$ is better able to suppress the slowly varying signals. We compared the output of 3 values of $\boldsymbol{\sigma}$ (0.1,1,10) and found that the noise in background is least when $\boldsymbol{\sigma}$=10.

● Comparing Box filters and Gaussian filters for Simple derivative

Here we use a simple derivative 0.5*[-1,0,1] to find the temporal evolution of the filtered image frames. We also calculate the standard deviation of first 'n' images for each case (n=20 for 'Office' and n=40). The values obtained are:

Office Dataset

Std Deviation (3x3) = 0.46695825576614064

Std Deviation (5x5) =0.32788602739066147

Std Deviation (Gaussian $\boldsymbol{\sigma}_{0.1}$) =0.9237176077465538

Std Deviation (Gaussian $\boldsymbol{\sigma}_{1}$) = 0.429215437505196

Std Deviation (Gaussian $\boldsymbol{\sigma}_{10}$)= 0.3278402470160502

Here, we obtain almost same std deviation in case of a 5x5 box filter and gaussian filter with $\boldsymbol{\sigma}$=10. They both reduce the noise equally and perform better than other filters. We can see that the masked image obtained in both cases is also the same.

- **Comparing Box filters and Gaussian filters for gaussian derivative**

Here we use a gaussian derivative of $\boldsymbol{\sigma}$=0.3 to find the temporal evolution of the filtered image frames. We also calculate the standard deviation of first 'n' images for each case (n=20 for 'Office' and n=40). The values of std deviation obtained are:

Office Dataset

Std Deviation (3x3) = 0.04011607389465371

Std Deviation (5x5) =0.02816847103869535

Std Deviation (Gaussian $\boldsymbol{\sigma}_{0.1}$) =0.07935596673273435

Std Deviation (Gaussian $\boldsymbol{\sigma}_{1}$) = 0.036873613422754865

Std Deviation (Gaussian $\boldsymbol{\sigma}_{10}$)= 0.028164538077090857

The std deviation is greatly reduced in case of using gaussian derivative to perform temporal derivative. It was much higher in case of prewitt derivative and simple derivative. Also, we have similar output of masked images for 5x5 box filter and gaussian filter with $\boldsymbol{\sigma}$=10.

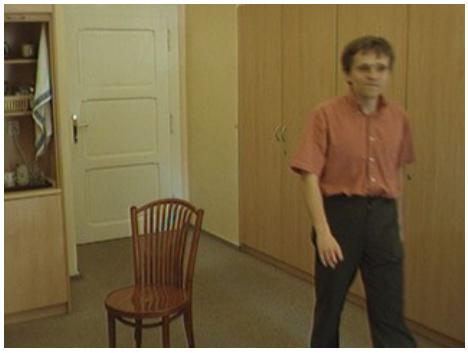
We can conclude that the values that make our algorithm run better are those that perform greater noise reduction and thus are able to suppress the background pixels better. By saying that our algorithm is performing better, we mean that the output masked image is showing more if the moving objects pixels as 1 and keeping stationary objects pixels as 0. It's important to note that increasing the size of filters or the value of $\boldsymbol{\sigma}$ indefinitely doesn't guarantee better results. We might miss some important small details that way. This has been discussed further in the report. Thus, depending upon our needs, we can filter the image with a balanced filter and perform the temporal derivative using the derivative that suits best to our application.

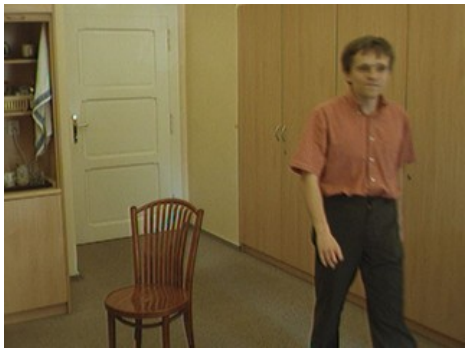
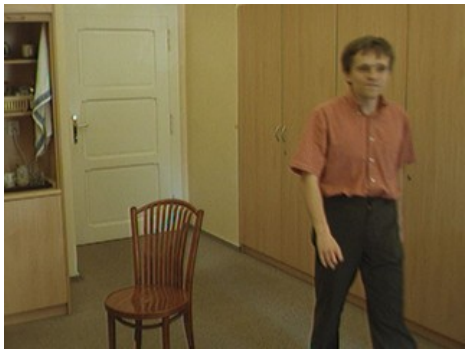
Different Algorithms for Thresholding

- **Maximum value of Temporal gradient:-** In this approach we estimate the maximum of the absolute values of temporal gradient using the stationary pictures and then use that maximum to threshold the values of temporal gradient.

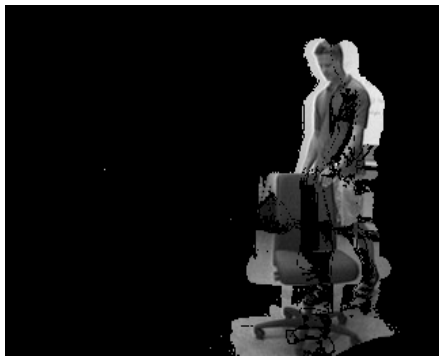
For the Office dataset we use the first 40 images that are stationary and for the RedChair dataset we use the first 20 images that are stationary to calculate the maximum absolute value of the temporal gradient. Below are the results using the maximum absolute value of temporal gradient: -

Office Dataset

Threshold	Filter	Original Image	Masked Image
14	Prewitt		
			

7	Simple	 	 
0.60	Gaussian $\boldsymbol{\sigma}$ = 0.3	 	 

RedChair Dataset

Threshold	Filter	Original Image	Masked Image
21	Prewitt		
			
10.5	Simple		

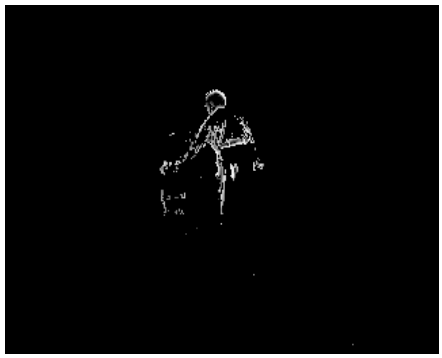
			
0.9	Gaussian $\boldsymbol{\sigma}$ = 0.3	 	 

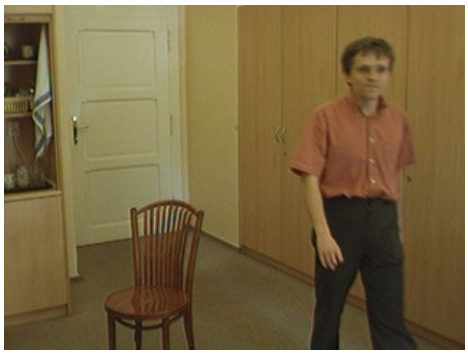
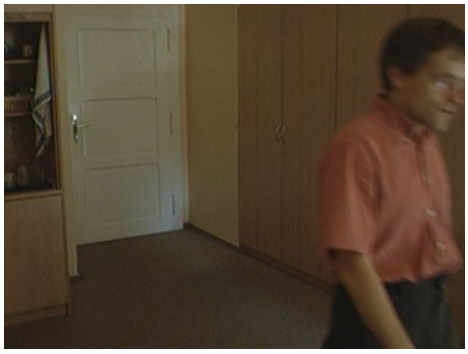
Fig: Output images upon using max value of temporal gradient as threshold

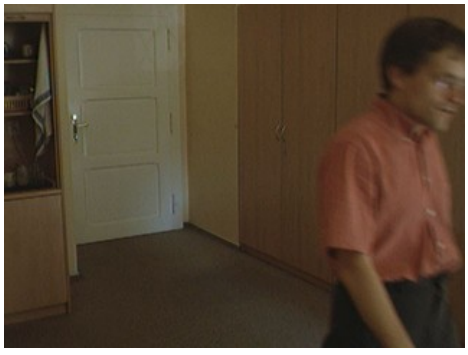
- **Using standard deviation of noise as threshold:-** In this approach we estimate the standard deviation of the temporal gradient using the stationary pictures which would essentially give us the noise in the temporal gradient of the images and then we use that standard deviation to threshold the values of temporal gradient. For the stationary pictures we know for sure that ideally the value of temporal gradient should be 0.

For the Office dataset we use the first 40 images that are stationary and for the RedChair dataset we use the first 20 images that are stationary to estimate the standard deviation of the temporal gradient in

stationary pictures. We are then using this standard deviation as a threshold for each image. Below are the results using the standard deviation of noise as threshold: -

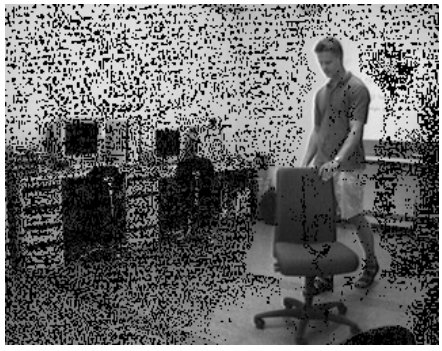
Office Dataset

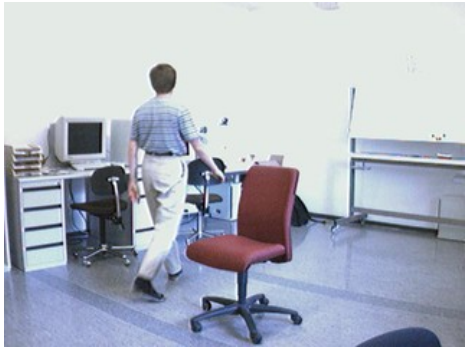
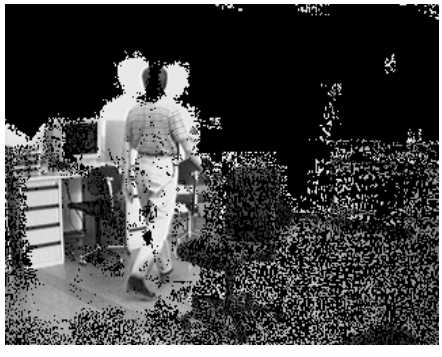
Threshold	Filter	Original Image	Masked Image
1.86	Prewitt		
			
0.93	Simple		

			
0.08	Gaussian σ = 0.3		
			
0.0653	Gaussian σ = 0.3 + 5X5 smoothing		

RedChair Dataset

Threshold	Filter	Original Image	Masked Image
3.905	Prewitt		
			

1.95	Simple	 	 
0.16774	Gaussian $\boldsymbol{\sigma} = 0.3$	 	 

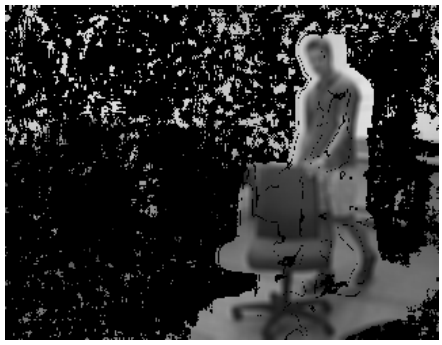
1.1389	Simple + Gaussian smoothing (5X5) $\boldsymbol{\sigma}$ = 15	 	 

Fig: Output images upon using std deviation of stationary frames as threshold

Observations

- **Comparing Maximum value of Temporal gradient to standard deviation of noise as threshold**

Both the methods that we have used to detect motion have their own pros and cons. For the maximum value threshold we usually lose small variations in the grayscale like - we get rid of the shadows while detecting the motion as we can see in the experimental results. The good part of the maximum value threshold is that we get very few false positives so we can use this algorithm in a case where we don't care about small changes in grayscale values and small details like shadows.

Whereas for the standard deviation threshold algorithm we can successfully track very small changes in the temporal gradient and we can successfully track shadows etc. A con of using this algorithm is that we get a lot of false positives while thresholding. Overall we feel that the standard deviation threshold algorithm works better in general and is able to successfully capture changes in small details as well as large gradient changes. If we use a smoothing filter to smooth the images and then use standard deviation threshold to compute the masked image we find that smoothing gets rid of false positives and results in an even better detection of motion as we can see above in the experimental results.

Conclusion

In the above report, we discussed how our algorithm helps in motion detection where most of the pixels in the frame belong to a stationary background and relatively small moving objects pass in front of the camera. We explained how our algorithm works and what each function does. In the experiments section, we displayed the results along with the images we got by using different temporal derivative filters. The images shown are representative of our ‘good’ and ‘bad’ results. Further, we compared the results obtained by filtering images with box filters and gaussian filters, and showed how varying these parameters changes the results of our motion detection algorithm. Finally, we discussed two different strategies of thresholding, using maximum value of temporal derivative as threshold and using the std deviation of the initial stationary images as threshold (std deviation of the noise). Both the strategies have their own pros and cons and we discussed where we could use each of them. We also tested both our strategies of thresholding on filtered images and observed how we could use smoothing filters to reduce the problem of false positives in case of std deviation thresholding.

Appendix

```
import cv2
import os
import numpy as np
import sys
np.set_printoptions(threshold=sys.maxsize)


class MotionDetection:
    def __init__(self, images_path: str, type_derivative_mask: str):
        self.images_path = images_path
        self.type_derivative_mask = type_derivative_mask

    def img_array(self):
        imgs_arr = []
        for img_name in os.listdir(self.images_path):
            img = cv2.imread(self.images_path + '\\' + img_name, 0)
            imgs_arr.append(np.asarray(img).astype(int))
        return imgs_arr

    def temporal_derivative(self, imgs_arr: list):
        i = 0
        temporal_derivative = []

        while i < len(imgs_arr):
            if 1 < i:
                temp_img_0 = imgs_arr[i - 2]
                temp_img_2 = imgs_arr[i]
                if self.type_derivative_mask == "prewitt":
                    temporal_derivative.append(np.subtract(temp_img_2, temp_img_0))
                elif self.type_derivative_mask == "simple":
                    temporal_derivative.append(0.5*np.subtract(temp_img_2, temp_img_0))
                else:
                    sigma = 0.3
                    x_0 = (-(-1) / (np.square(sigma))) * np.exp(-(np.square(-1)) / (2 * np.square(sigma)))
                    x_2 = (-1 / (np.square(sigma))) * np.exp(-(np.square(1)) / (2 * np.square(sigma)))
                    temporal_derivative.append(np.add(x_2 * temp_img_2, x_0 * temp_img_0))
            i += 1
```

```
        return temporal_derivative

    @staticmethod
    def filtering(filter_type: str, imgs_arr: list):
        filtered_imgs = []
        if filter_type == '3x3':
            kernel1 = np.ones((3, 3), np.float32)/9
            for i in imgs_arr:
                filtered_imgs.append(cv2.filter2D(i.astype(np.uint8), ddepth=-1, kernel=kernel1).astype(int))
        elif filter_type == '5x5':
            kernel2 = np.ones((5, 5), np.float32)/25
            for i in imgs_arr:
                filtered_imgs.append(cv2.filter2D(i.astype(np.uint8), ddepth=-1, kernel=kernel2).astype(int))
        else:
            for i in imgs_arr:
                filtered_imgs.append((cv2.GaussianBlur(i.astype(np.uint8), (5, 5), 15)).astype(int))
        return filtered_imgs

    @staticmethod
    def thresholding(temporal_derivative: list, type_of_thresh: str):

        if type_of_thresh == 'max':
            mx = 0
            mn = 0
            for td in temporal_derivative:
                if np.max(td) > mx:
                    mx = np.max(td)
                if np.min(td) < mn:
                    mn = np.min(td)
                return max(abs(mx), abs(mn))

        elif type_of_thresh == 'std':
            ideal_thresh = np.zeros(np.shape(temporal_derivative[0]))
            var_arr = np.zeros(np.shape(temporal_derivative[0]))

            for img in temporal_derivative:
                var_arr = var_arr + np.square(ideal_thresh - img)
            std_dev_arr = np.sqrt(var_arr * 1 / (len(temporal_derivative) - 1))
            average_std_dev = np.mean(std_dev_arr)
            return average_std_dev
```

```
    @staticmethod
    def combine_mask(temporal_derivative: list, imgs_arr: list, threshold: float):
        masked_imgs = []
        imgs_arr.pop(0)
        imgs_arr.pop(-1)
        i = 0
        for temp_der_i in temporal_derivative:
            temporal_derivative = np.where(abs(temp_der_i) > threshold, 1, 0)
            masked_imgs.append(np.multiply(imgs_arr[i].astype(np.uint8), temporal_derivative.astype(np.uint8)))
            i += 1
        return masked_imgs


if __name__ == "__main__":
    md = MotionDetection(r'D:\NEU\3 Sem\Computer Vision\Project 1\RedChair', "simple")
    array_of_images = md.img_array()
    filter_images = md.filtering('gauss', array_of_images)
    temp_derivative = md.temporal_derivative(filter_images)
    thresh = md.thresholding(temp_derivative[:20], "std")
    masked_images = md.combine_mask(temp_derivative, filter_images, thresh)

    for ind in range(len(masked_images)):
        cv2.imwrite(f"Threshold output\\image{ind}.png", masked_images[ind])
```