Motion Detection Using Simple Image Filtering

Sahasrajit Anantharamakrishnan[1], **Eli MacColl**[2]

[1]Department of Electrical and Computer Engineering, anantharamakrishn.sa@northeastern.edu
[2]Department of Electrical and Computer Engineering, maccoll.e@northeastern.edu

February 20, 2023

Abstract

This project aims to explore a straightforward method for detecting motion in image sequences captured using a stationary camera. The majority of the pixels in these images correspond to a static background with objects occasionally passing in front of the camera. In this scenario, when observing the gray-scaled version of these images, the intensity values of pixels remain the same or slightly fluctuate over time except when a moving object crosses the frame. The pixels of the moving object in the foreground replace the background intensity with its own. Consequently, identifying significant changes in the temporal evolution of pixel values is a fundamental method to detect the presence of moving objects in the scene. We can effectively track and identify moving objects in image sequences by analyzing these changes.

1. Introduction

Motion detection is a fundamental task in computer vision that involves detecting and tracking moving objects in a sequence of images. It has numerous applications, including surveillance, traffic monitoring, robotics, and many more. Motion detection typically involves analyzing the differences between consecutive frames in a sequence of images to identify areas of motion. This process can be accomplished using a variety of techniques, including background subtraction, optical flow, and feature tracking. The choice of technique depends on the specific application, the characteristics of the images, and the available computational resources.

In recent years, advances in machine learning and deep learning have led to the development of novel techniques for motion detection that leverage the power of neural networks to achieve state-of-the-art performance. This has opened up new possibilities for motion detection in computer vision and expanded its potential applications. In this context, motion detection continues to be an active area of research, and its significance is expected to grow with the increasing demand for automated visual systems that can detect and track objects in real time.

In this project, we implement background subtraction, the simplest technique for detecting motion in a sequence of images. This process involves computing the difference between the current frame of the sequence and the previous frame to identify changes in the foreground. Background subtraction is typically a fast and cheap method in terms of computational resources compared to other motion detection techniques. That being said, it can be significantly affected by several factors, including changes in lighting, and shadows, gradual changes occurring in the background, and inherit noise generated during image capture that reduce the accuracy of the results.

2. Algorithms

Throughout this project, we employed fundamental image processing techniques, such as convolution and thresholding, to enhance the accuracy of the motion detection process.

Thresholding is used to segment images into distinct regions based on intensity values. A threshold value is selected to be compared against every pixel in the image. Pixels with intensity values below the threshold are designated to one region, and pixels with values above the threshold are assigned to the other. Applying a threshold on a gray-scale image where pixel intensity is represented by 256 shades of gray $\in [0, 255]$ can be used to obtain a mask that clearly defines these regions. In this case, a binary mask can be generated to represent the pixels of moving objects in the foreground.

The equation of the thresholding function, $T(x)$, is given by:

$$T(x) = \begin{cases} y_1, & \text{if } x < t_1 \\ y_2, & \text{if } t_1 < x \leq t_2 \\ y_3, & \text{if } x > t_2 \end{cases}$$

Here,

- $T(x)$ is the thresholding function.
- x is the intensity value at a given pixel location.
- t_1 and t_2 are the lower and upper threshold values respectively.
- y_1, y_2 and y_3 are the outputs of the thresholding function.

Similarly, for binary thresholding:

$$T_B(x) = \begin{cases} y_1, & \text{if } x < t_1 \\ y_2, & \text{if } x \geq t_1 \end{cases}$$

Convolution is a technique that involves applying a kernel to an image to perform operations such as smoothing, sharpening, edge detection, and feature extraction. The kernel is placed at each pixel in the image and convolution is performed by multiplying the kernel's values the with corresponding pixel values in the image, summing the results, and assigning the sum to the center pixel. Repeating this process at every pixel in an image produces a new image with some effect. The choice of kernel depends on the desired outcome of the resulting image. In our experiments for detecting motion, We applied convolution for edge detection and smoothing to reduce noise.

Convolution of an $M \times N$ image, $f(x, y)$, with a filter $w(x, y)$ whose size is $m \times n$, is given by:

$$g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) \cdot f(x - s, y - t)$$

Where,

$$a = \frac{(m-1)}{2}; \quad b = \frac{(n-1)}{2}$$

Here

- $g(x, y)$ is the image obtained after convolution.
- $f(x, y)$ is the original image of size $M \times N$.
- $w(x, y)$ is the filter whose size is $m \times n$.

- Finally, x, y are pixel coordinates.

We can also use the formula for cross-correlation instead of convolution, since the filter is symmetric, there is no difference between convolution and cross-correlation. The formula for cross-correlation is:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) \cdot f(x+s, y+t)$$

Here, a and b have the same definition as convolution.

3. Experiments

The programs used to carry out this project operate according to the following basic outline, this is used as a "baseline" or "standard" result for the next set of experiments:

(a) Read the given sequence of images.

(b) Convert them to grayscale.

(c) Apply a 1-D Differential operator at each pixel in order to compute a temporal derivative.

(d) Threshold the absolute values of the derivatives to create a binary mask of the moving objects.

(e) Combine the mask with the original frame to display the resulting image.

On the fundamental outline mentioned above, we investigated variations of this process that were implemented across experiments in an effort to improve upon and better understand the results. These were:

(i) Using a simple $\frac{1}{2}\begin{bmatrix}-1 & 0 & 1\end{bmatrix}$ filter and a 1-D Gaussian derivative with a user-defined standard deviation `tsigma` (σ_t) for the temporal derivative filter. We varied the values of σ_t to get different results.

(ii) Prior to using the temporal derivative filter in the baseline, we added a 2D spatial smoothing filter to the frames. Filters that were used for spatial smoothing:

   - 3×3 box filter.
   - 5×5 box filter.
   - 2D Gaussian filter with a user-defined standard deviation `ssigma` (σ_s).

   Here we experimented with different values of σ_s to get different results.

(iii) Finally, we experimented with various threshold values to get a mask that works well for both datasets.

3.1. Office Chair Dataset

The dataset captures the movement of a man who enters the camera frame in a somewhat parallel direction to its optical axis. After exiting the frame, he reappears, carrying a chair with him, which he places in the foreground and then leaves the frame. Several seconds later, he returns to turn on the overhead light and exits the frame again. Finally, he returns to retrieve the chair and leaves with it.

3.2. Red Chair Dataset

The movement of a man is recorded in the dataset as he walks from left to right across the frame, perpendicular to the camera's optical axis. Subsequently, he enters the frame on the right while pushing a chair, which he places in the center of the frame and exits on the left. Following this, he walks back across the frame to the right to turn on a bright light (which is outside of the frame and perpendicular to the background wall) before entering the frame again on the right and walking toward the left side of the frame. After a brief pause, he heads towards the computer situated in the background to stop the recording.

4. Results and Discussion

The images displayed in the following results represent a small subset of our outputs. These select images aim to best demonstrate our overall results since we cannot include them all for the sake of brevity.

4.1. Baseline

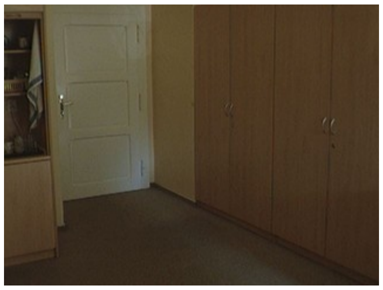
In the baseline experiment, we have successfully detected motion as you can see from Figures 1 and 2.

In both datasets, we can see that the masks are non-zero only when there is motion in the frame, meaning the pixels are in motion. This is clearly exhibited in the 'Red chair' dataset where, in one of the frames, Figure 2d the left leg is moved whilst the right leg is stationary. The filter also correctly identifies this and does not detect the right leg.

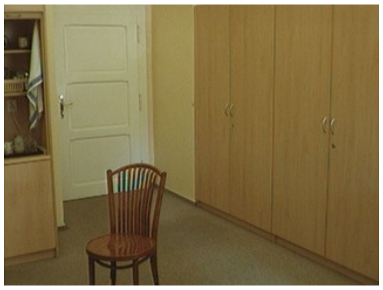
To demonstrate the accuracy of this motion detector we have included frames with and without a motion from both datasets. As shown in Figures 1a to 1c and Figures 2a and 2b, there are no moving objects in the frame, and our resulting images correctly show no mask over any pixels in those images.

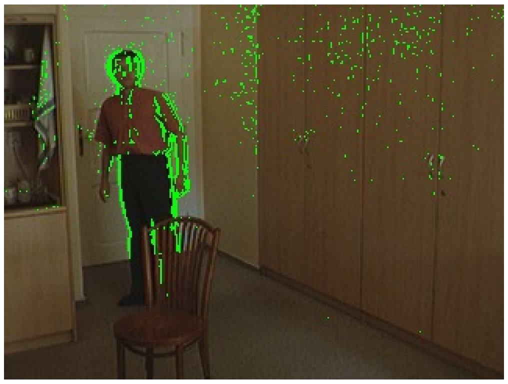
Background subtraction works on the principle that the background remains relatively constant, so a sudden change in lighting conditions has a noticeable impact on the results. In Figure 2c, when the lights are turned on there is a sharp increase in the intensity values of all the pixels in the image. In the short sequence of frames where the lights are being turned on, this rudimentary form of motion detection triggers a false positive for a number of background pixels. A similar effect can be seen from the false positives on the walls in images from the 'Office' dataset and is exhibited in Figure 1d.

In Figure 2e, there is a "ghost" image of the previous frame that is a consequence of subtracting the current frame from the previous frame. This effect is also present in Figure 1e, but it is less apparent because the motion is towards the camera, so the current frame is "in front of" the previous frame.


(a)

(b)


(c)


(d)

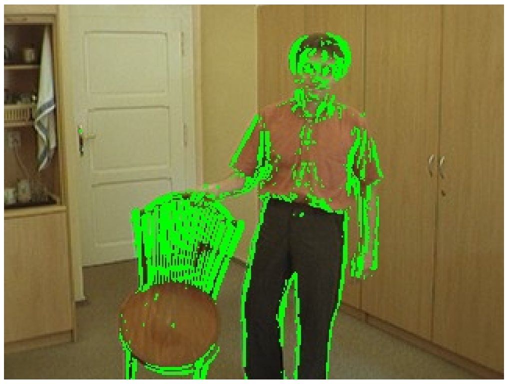

(e)

Figure 1. Office dataset - Baseline results.

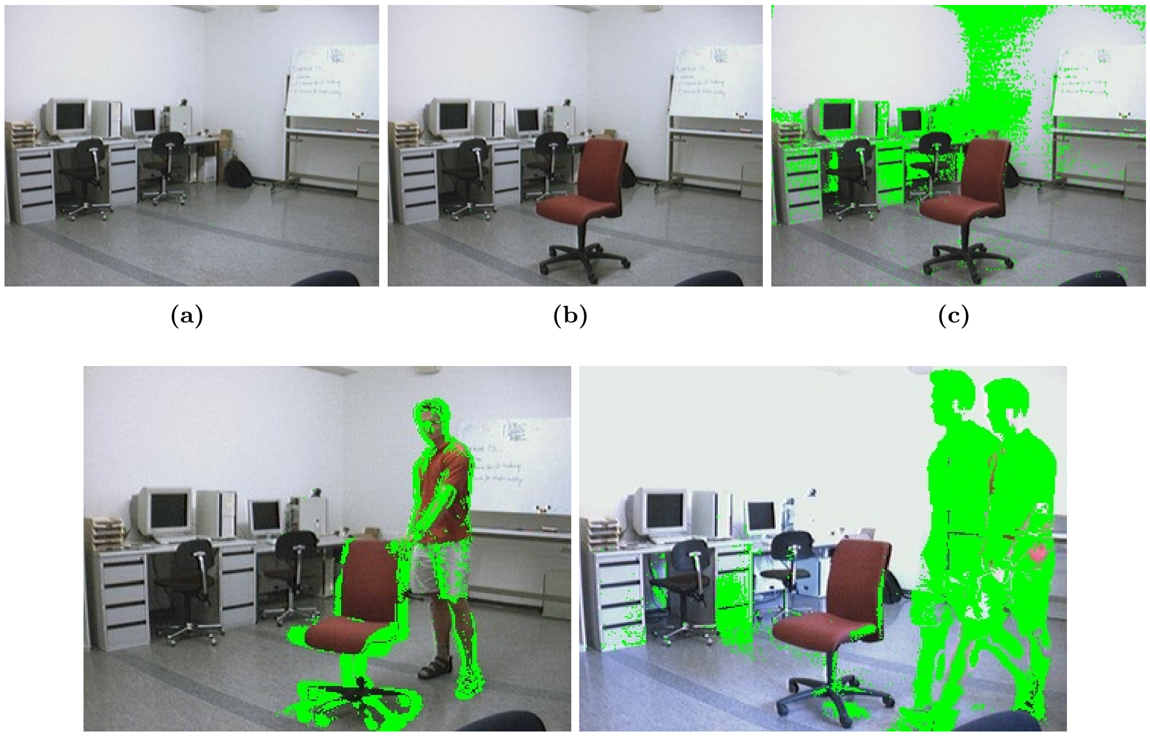

(a) **(b)** **(c)** **(d)** **(e)**

Figure 2. Red Chair dataset - Baseline results.

4.2. Experiment 1: $\frac{1}{2}\begin{bmatrix}-1 & 0 & 1\end{bmatrix}$ filter and a 1-D Gaussian derivative with σ_t

In experiment 1, we are using a vertical edge-detecting filter and a 1D Gaussian smoothing filter as opposed to a motion detector. A vertical edge detector is used to detect the vertical edges in the image, not the motion hence, even with the best σ_t value it detects stationary objects when the objective is to detect moving objects. We varied the value of σ_t for the Gaussian filter to observe and compare the results.

The metrics by which we define a certain σ_t value producing a "better" resulting image are based on the ratio of signal to noise. The optimal result is an image with no extraneous noise, and all of the vertical edges properly detected. In practice, we experimented with different σ_t and examined the output image to determine the efficacy of the filter. With a lower $\sigma_t = 0.1$, there was very little noise, but many edges were not detected when they should have been as shown in Figures 3a and 4a. With a larger value of $\sigma_t = 1$, edges are properly detected, but there is also a lot of noise in the image as seen in Figures 3c and 4c. Through

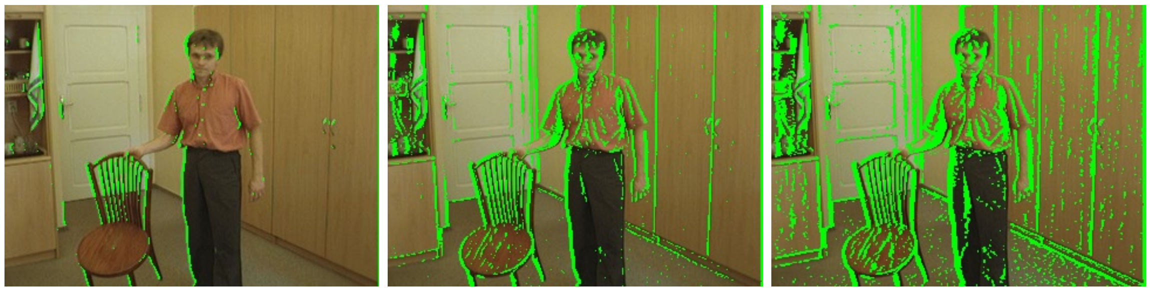
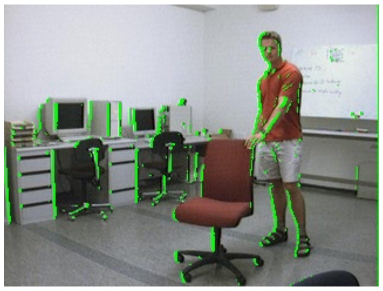
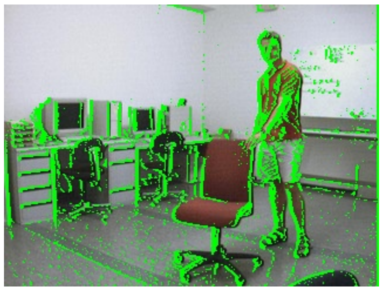

(a) $\sigma_t = 0.1$ **(b)** $\sigma_t = 0.6$ **(c)** $\sigma_t = 1.0$

Figure 3. Office dataset - Experiment 1


(a) $\sigma_t = 0.1$


(b) $\sigma_t = 0.6$

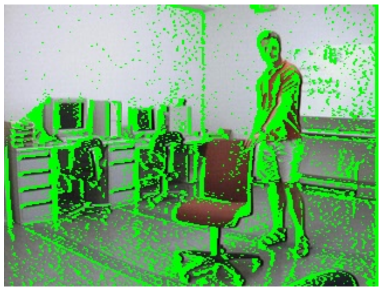

(c) $\sigma_t = 1.0$

Figure 4. Red Chair dataset - Experiment 1

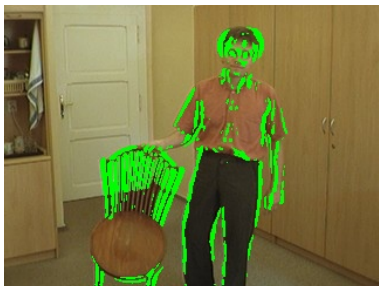
this trial process, we found that a value around $\sigma_t = 0.6$ produced the "best" results for both datasets, Figures 3b and 4b. Many of the edges are correctly detected with only a small degree of noise.

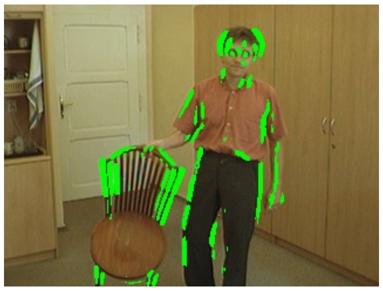
4.3. Experiment 2: 2D spatial smoothing filter before baseline

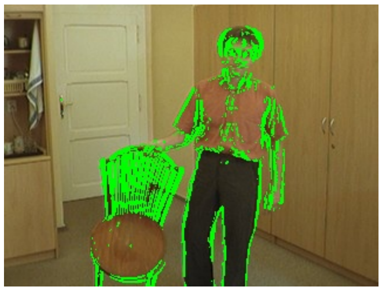
Experiment 2 involved applying 2D spatial smoothing filters to the image frames prior to applying the temporal derivative filter to detect motion. A smoothing filter works to reduce the amount of noise in the image by replacing the value of each pixel with an average of its neighbors. This results in a smoother and more blurred version of the original image with less sharp transitions, such as edges. Figures 5 and 6 exhibit the effect that various smoothing filters had on the temporal derivative filter for motion detection.

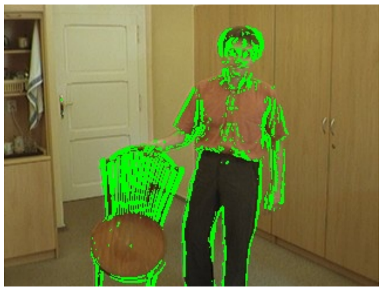
In Figures 5a and 6a, a 3×3 box filter was applied to each image before computing the temporal derivative. The image generated by applying the 3×3 box filter resulted in blurrier edges, which in turn produced larger outlines around objects in motion in the output image. This effect is accentuated slightly in Figures 5b and 6b where a larger box filter of size 5×5 was applied.

We also tried applying 2D Gaussian filters with various values of σ_s to produce different amounts of


(a) 3×3 box filter.


(b) 5×5 box filter.


(c) $\sigma_s = 0.1$


(d) $\sigma_s = 0.2$

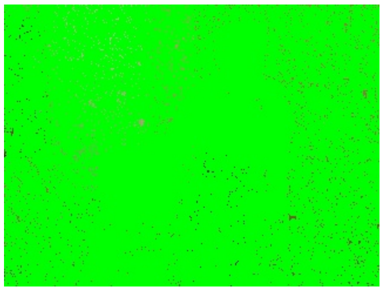
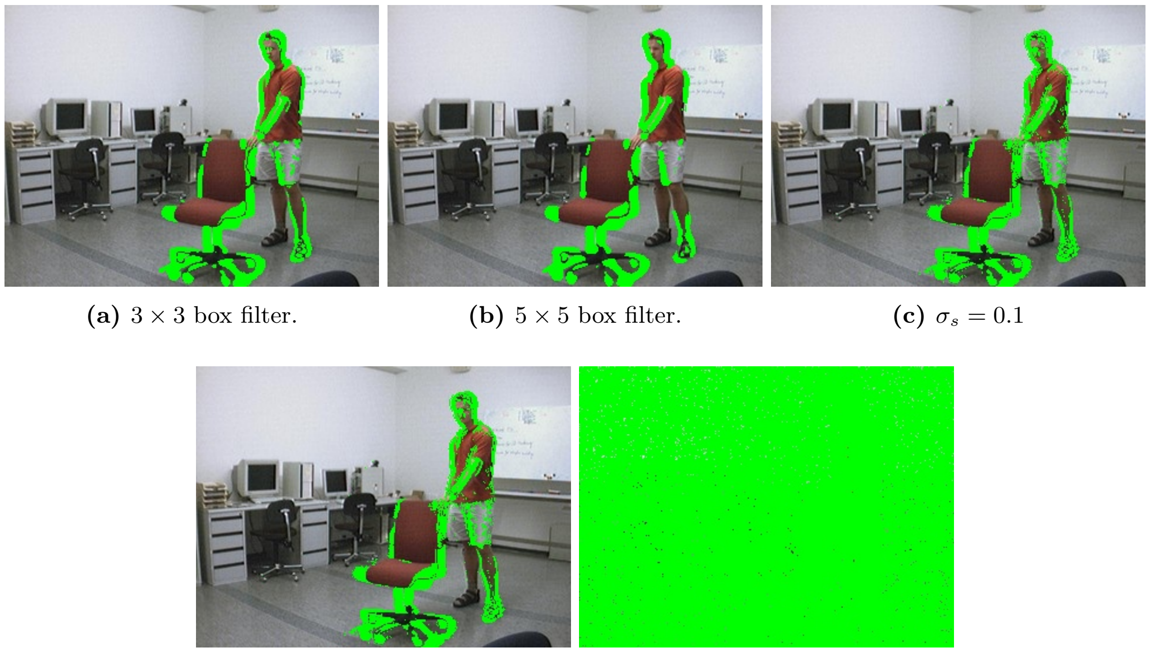

(e) $\sigma_s = 0.25$

Figure 5. Office dataset - Experiment 2



(a) 3×3 box filter. **(b)** 5×5 box filter. **(c)** $\sigma_s = 0.1$

(d) $\sigma_s = 0.2$ **(e)** $\sigma_s = 0.25$

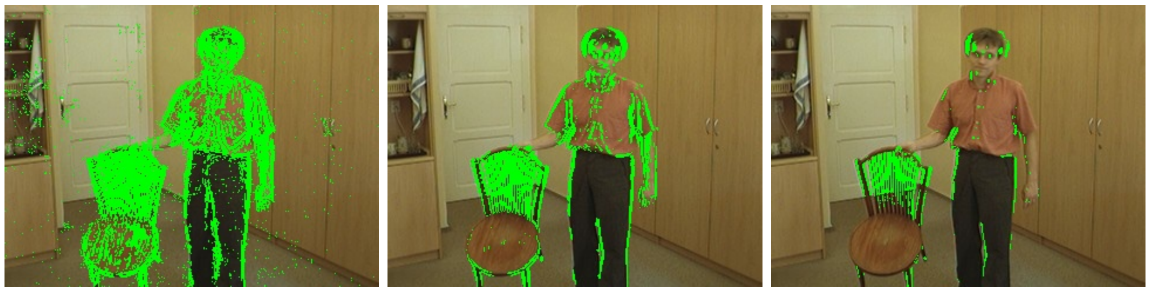
Figure 6. Red Chair dataset - Experiment 2

smoothing on the images. To compute the Gaussian filters, we used $M = \lceil 5\sigma_s \rceil$ where M is the size of the 2D Gaussian Filter (size $= M \times M$). We found that when the size of the Gaussian filter was any larger than 1×1, it was actually detrimental to the outcome because the original image was extensively smoothed to a degree where objects in the image could not be differentiated using pixel intensity as seen in Figures 5e and 6e with $\sigma_s = 0.25$ which produces a Gaussian of size 2×2. In turn, computing the temporal derivative becomes ineffective for detecting motion across the sequence of images. For smaller values of $\sigma_s = 0.2$ and $\sigma_s = 0.1$ as show in Figures 5c, 5d, 6c and 6d, according to the equation above, they both produce 1×1 Gaussian filters, effectively not altering the original image.

4.4. Experiment 3: Threshold Values

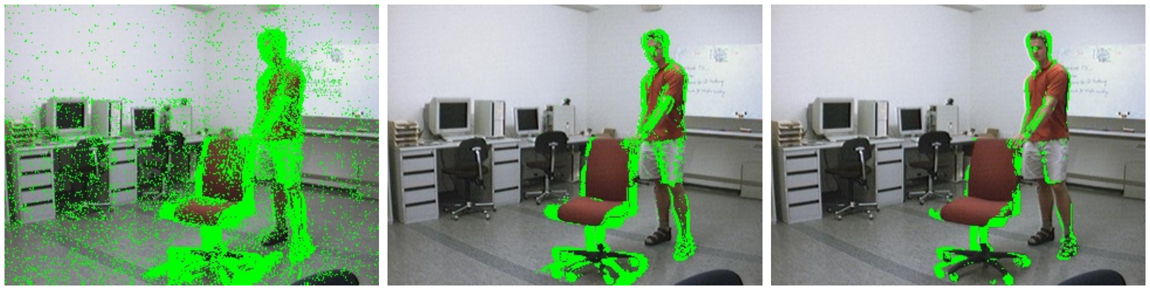
In Experiment 3, using our baseline algorithm, we varied the threshold value used to generate the binary mask that defines which pixels are "in motion". After computing the temporal derivative, we threshold the results to generate a mask of 1s and 0s where a 1 represents a "moving" pixel. In practice, a 1 in the mask is represented by an intensity value of 255. By altering the selected threshold value we are defining the minimum intensity value of a pixel required to be considered part of a moving object. A lower threshold value results in more "moving" pixels whereas a higher threshold value leads to less. The optimal threshold value is one where all motion in the frame is detected and there are no false positives or false negatives.

In order to determine a good threshold value for the two datasets, we started small and increased it incrementally while examining the resulting images each time. We continued this processes until we started noticing movement going undetected, indicating that the threshold was too high. For a low threshold value, such as $t_1 = 5$, the motion in the image sequence is detected, but there are a lot of false positives as shown in Figures 7a and 8a. When we reached a higher threshold value of $t_1 = 50$, we started noticing that pixels we knew to be "in motion" (false negatives) were no longer being detected which can be seen in Figures 7c and 8c. This process ultimately led us to find a threshold value of $t_1 = 20$ to be effective for both datasets. As exhibited in Figures 7b and 8b, the motion in the image is clearly detected with little to no activity detected anywhere else in the image, as expected.



(a) $t_1 = 5$ **(b)** $t_1 = 20$ **(c)** $t_1 = 50$

Figure 7. Office dataset - Experiment 3



(a) $t_1 = 5$ **(b)** $t_1 = 20$ **(c)** $t_1 = 50$

Figure 8. Red Chair dataset - Experiment 3

5. Conclusion

In conclusion, the project "Motion Detection Using Simple Image Filtering" demonstrates a practical application of image processing techniques to detect motion in video sequences. The project shows how basic image filtering operations, such as thresholding, convolution, and differencing, can be used to highlight areas of change between consecutive frames and identify moving objects in a scene.

The performance of the motion detection algorithm depends on several factors, such as the choice of filter parameters and the quality of the input images, and characteristics such as lighting. The project provides examples of how to fine-tune the filter parameters to improve the accuracy of motion detection and reduce false positives.

Overall, the project shows how simple image filtering techniques can be a powerful tool for motion detection. By applying these techniques to a sequence of images, it is possible to extract useful information about the movement of objects and analyze the dynamics of complex systems.

Appendix - Code

A. Baseline

```python
import cv2
import numpy as np
import os
import scipy.signal as sp
import matplotlib.pyplot as plt
from math import ceil, floor

Z = 0
DIR = "RedChair" if Z else "Office"
images = [
    os.path.join(DIR, f)
    for f in sorted(os.listdir(DIR))
    if f.endswith(".jpg")
]

colour = [cv2.imread(f) for f in images]
gray = [cv2.imread(f, 0) for f in images]


ind = 0
for i in range(len(gray) - 1):
    diff = cv2.absdiff(gray[i], gray[i + 1])

    # Thresh 20
    thresh = cv2.threshold(diff, 20, 255, cv2.THRESH_BINARY)[1]

    col = colour[i].copy()
    col[thresh == 255] = [0, 255, 0]

    cv2.imshow("image", col)
    cv2.imshow("thresh", thresh)

    # cv2.imwrite(f"output/{DIR}/diff/image_{ind}.jpg", diff)
    # cv2.imwrite(f"output/{DIR}/diffThresh/image_{ind}.jpg", thresh)
    # cv2.imwrite(f"output/{DIR}/diffThreshColor/image_{ind}.jpg", col)
    # ind += 1

    if cv2.waitKey(20) & 0xFF == ord("q"):
        break
```

B. Experiment 1: $\frac{1}{2}\begin{bmatrix}-1 & 0 & 1\end{bmatrix}$ filter and a 1-D Gaussian derivative with σ_t

```python
import cv2
import numpy as np
import os
import scipy.signal as sp
import matplotlib.pyplot as plt
from math import ceil, floor

def createGaussian(sigma):
    size = ceil(5 * sigma)
    s2 = sigma * sigma
    mask = np.zeros((size, size))

    low = floor(size/2)
    high = ceil(size/2)
    for x in range(-low, high):
        for y in range(-low, high):
            mask[x+low, y+low] = np.exp((-x**2 - y**2) / (2 * s2))
    mask /= np.min(mask)
    mask = np.round(mask)
    return mask


Z = 0
DIR = "RedChair" if Z else "Office"
images = [
    os.path.join(DIR, f)
    for f in sorted(os.listdir(DIR))
    if f.endswith(".jpg")
]

colour = [cv2.imread(f) for f in images]
gray = [cv2.imread(f, 0) for f in images]

p1DH = np.array([[-0.5, 0, 0.5]])

tSigma = 0.6
g1DH = np.array([createGaussian(tSigma)[0]])

ind = 0
for i in range(len(gray)):
    diff = sp.convolve2d(gray[i], p1DH, mode="same", boundary="fill", fillvalue=0)
    diff = sp.convolve2d(diff, g1DH, mode="same", boundary="fill", fillvalue=0)

    # Thresh 20
    thresh = cv2.threshold(diff, 20, 255, cv2.THRESH_BINARY)[1]

    col = colour[i].copy()
    col[thresh == 255] = [0, 255, 0]

    cv2.imshow("image", col)
    cv2.imshow("thresh", thresh)

    # cv2.imwrite(f"output/2bi/{DIR}/sigma0_1/thresh/image_{ind}.jpg", thresh)
    # cv2.imwrite(f"output/2bi/{DIR}/sigma0_1/color/image_{ind}.jpg", col)
    # ind += 1

    if cv2.waitKey(20) & 0xFF == ord("q"):
        break
```

C. Experiment 2: 2D spatial smoothing filter before baseline

```python
import cv2
import numpy as np
import os
import scipy.signal as sp
import matplotlib.pyplot as plt
from math import ceil, floor

```

```python
def createGaussian(sigma):
    size = ceil(5 * sigma)
    s2 = sigma * sigma
    mask = np.zeros((size, size))

    low = floor(size/2)
    high = ceil(size/2)
    for x in range(-low, high):
        for y in range(-low, high):
            mask[x+low, y+low] = np.exp((-x**2 - y**2) / (2 * s2))
    mask /= np.min(mask)
    mask = np.round(mask)
    return mask

Z = 0
DIR = "RedChair" if Z else "Office"
images = [
    os.path.join(DIR, f)
    for f in sorted(os.listdir(DIR))
    if f.endswith(".jpg")
]

colour = [cv2.imread(f) for f in images]
gray = [cv2.imread(f, 0) for f in images]

sSigma = 0.25
g2D = np.array(createGaussian(sSigma))

# size = 3
# size = 5
# box = np.ones((size,size)) * (1/(size ** 2))

gray = [sp.convolve2d(i, g2D, mode="same", boundary="fill", fillvalue=0) for i in gray]

ind = 0
for i in range(len(gray) - 1):
    diff = cv2.absdiff(gray[i], gray[i + 1])

    # Thresh 20
    thresh = cv2.threshold(diff, 20, 255, cv2.THRESH_BINARY)[1]

    col = colour[i].copy()
    col[thresh == 255] = [0, 255, 0]

    cv2.imshow("image", col)
    cv2.imshow("thresh", thresh)

    cv2.imwrite(f"output/2bii/{DIR}/gauss/sigma0_25/thresh/image_{ind}.jpg", thresh)
    cv2.imwrite(f"output/2bii/{DIR}/gauss/sigma0_25/color/image_{ind}.jpg", col)
    ind += 1

    if cv2.waitKey(60 if Z else 30) & 0xFF == ord("q"):
        break
```

D. Experiment 3: Threshold Values

```python
import cv2
import numpy as np
import os
import scipy.signal as sp
import matplotlib.pyplot as plt
from math import ceil, floor

Z = 0
DIR = "RedChair" if Z else "Office"
images = [
    os.path.join(DIR, f)
    for f in sorted(os.listdir(DIR))
    if f.endswith(".jpg")
]

colour = [cv2.imread(f) for f in images]
gray = [cv2.imread(f, 0) for f in images]


lowThresh = 15
ind = 0
for i in range(len(gray) - 1):
    diff = cv2.absdiff(gray[i], gray[i + 1])

    thresh = cv2.threshold(diff, lowThresh, 255, cv2.THRESH_BINARY)[1]

    col = colour[i].copy()
    col[thresh == 255] = [0, 255, 0]

    cv2.imshow("image", col)
    cv2.imshow("thresh", thresh)

    # cv2.imwrite(f"output/2biii/{DIR}/thresh_{lowThresh}/thresh/image_{ind}.jpg", thresh)
    # cv2.imwrite(f"output/2biii/{DIR}/thresh_{lowThresh}/color/image_{ind}.jpg", col)
    # ind += 1

    if cv2.waitKey(60 if Z else 30) & 0xFF == ord("q"):
        break
```