Algorithmic Foundations of Robotics and AI

LAB 3: Image Processing

Group 13
Michael Krone — Felix Coy — Peter Kruse
2514618 — 2505497 — 2596087
University of Turku

September 29, 2025

Task 1

```
python3 task1.py
```

Task 1 was straightforward, since the required functions are provided directly by OpenCV. The main work was to configure the drawing functions, add text, and place shapes on the image. The processed image was then saved into the processed_images folder for documentation, and it is also displayed below.

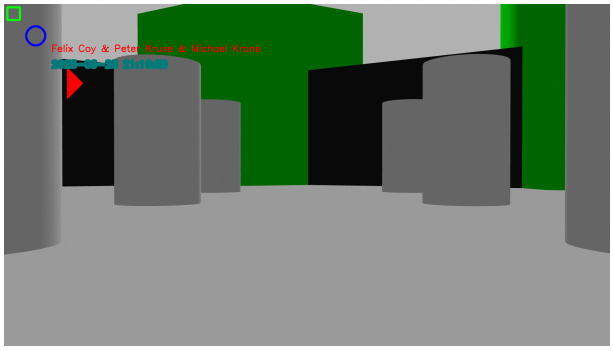


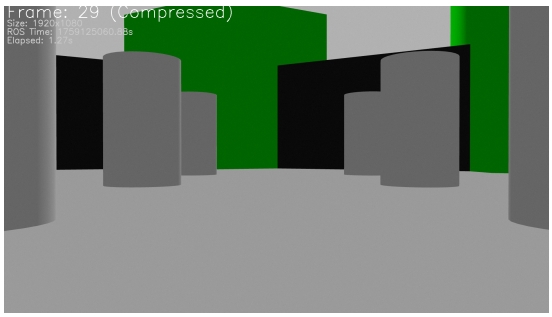
Figure 1: Task 1 Results

Task 2

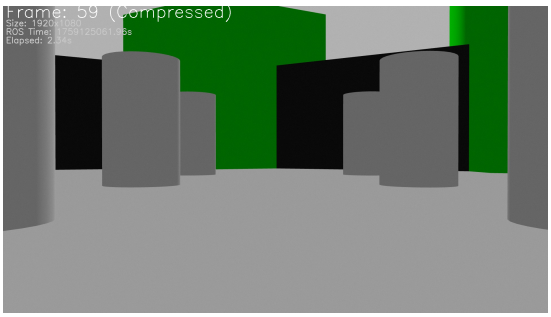
```
# Terminal A
ros2 bag play path/to/your/rosbag

# Terminal 2: Run your script
python3 task2.py
```

In Task 2 the processing was extended to a video stream from a rosbag. The compressed image messages first had to be converted into a usable image format before further steps. The same type of annotations as in Task 1 were then added directly on the incoming frames in real time. In addition, every 30th frame was saved automatically into the subfolder processed_images/task2 to record sample results from the stream.



(a) Picture 1



(b) Picture 2

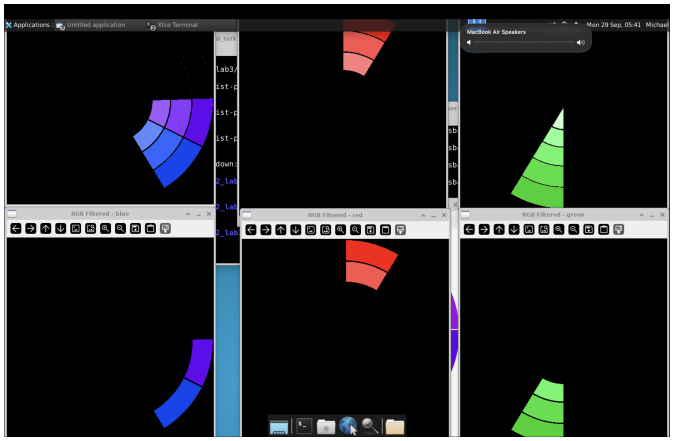
Figure 2: Task 2 Results

Task 3

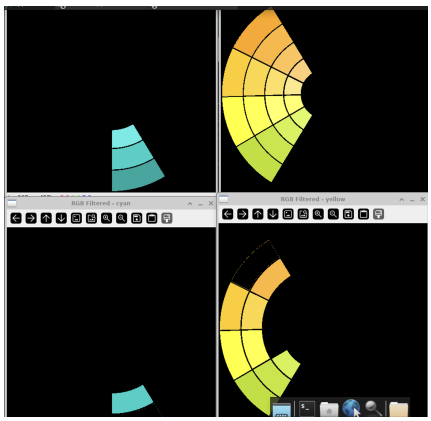
```
# Terminal A
python3 task3.py
```

In this task the goal was to set ranges for different colors, create masks with upper and lower bounds, and apply them to the image. Besides red, green, and blue, two more colors (cyan and yellow) were tested. The results showed that HSV filtering gives cleaner color detection, while RGB is more sensitive to brightness and noise.

I found RGB filtering the easiest to work with, since you only need to check the three channels directly and it feels very straightforward. But for real life tasks HSV is much more useful, because it separates the actual color (hue) from brightness and saturation. For example, if you want to track a red ball indoors and outdoors, the brightness will change a lot but the hue stays the same, so HSV filtering works more reliably. RGB would often fail in such cases because shadows or lighting easily change the values in all three channels.



(a) blue, red, green



(b) cyan, yellow

Figure 3: Task 3 Results

Task 4

```
# Terminal A
ros2 bag play path/to/your/lab3_2

# Termonal B
python3 task4.py
```

In Task 4 the idea was to take the filters from Task 3 and apply them to a live video stream from a rosbag. This was more challenging because the filtering had to be done frame by frame and the output updated in real time. The results were displayed in a 2×2 grid for easier visualization. Since HSV filtering is more stable in real-life situations and less affected by lighting changes, this approach was chosen here as well.

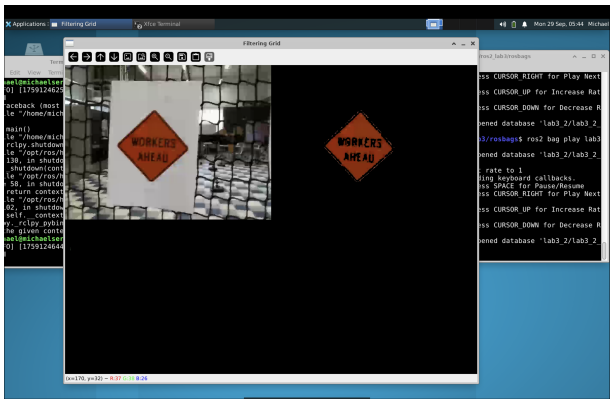


Figure 4: Task 4 Results

Task 5

```
# Terminal A
python3 task5.py
```

In Task 5 the challenge was to implement convolution manually and apply different blur filters to an image. A simple box filter with different kernel sizes (3×3, 5×5, 7×7) was tested as well as a Gaussian blur created with a custom kernel. To handle the borders of the image, zero padding was used, meaning the outside of the image is filled with black pixels during convolution. This way the output has the same size as the input image.

How does the kernel size affect the blur intensity and why?

The kernel size directly affects the blur intensity. A larger kernel results in stronger blur because it averages pixel values over a wider area. This incorporates more distant pixel information, leading to greater smoothing and loss of fine detail.

How do box blur and Gaussian blur differ in visual quality? Point to examples in your results

Gaussian Blur produces a smoother, more natural blur due to its weighted kernel, which applies greater influence to central pixels, resulting in smoother transitions. Box Blur uses uniform averaging, which can lead to harsher transitions and potentially a less smooth appearance compared to Gaussian blur.

Task 6

```
# Terminal A
python3 task6.py
```

In Task 6 the task was to define a kernel and apply it over the image using a convolution loop, storing the result in a new image. While the PDF mentioned a 3×3 kernel, the provided Python file already used a 5×5 kernel, so this size was chosen here as well. The main goal of this task is edge detection. After applying kernels in the x- and y-directions (for example with Sobel filters), the results are combined into a single image. This is done

(a) 3x3 Kernel

(b) 5x5 Kernel

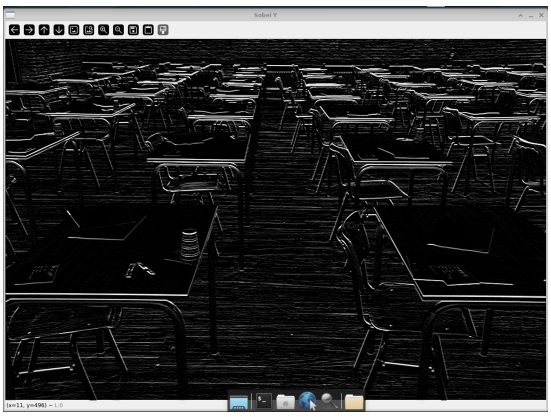
(c) 7x7 Kernel

(d) Gaussian Blur

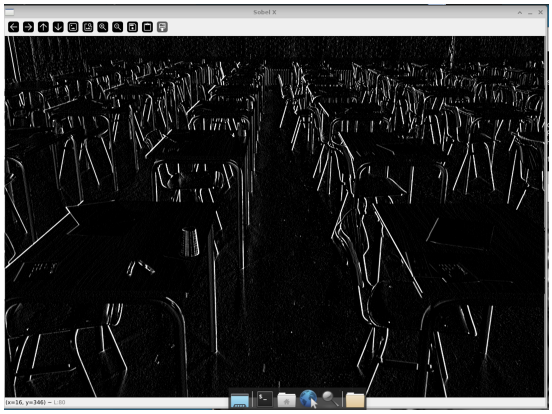
Figure 5: Different Kernel and Blur function

by calculating the magnitude, which means taking the strength of the edge regardless of its direction.

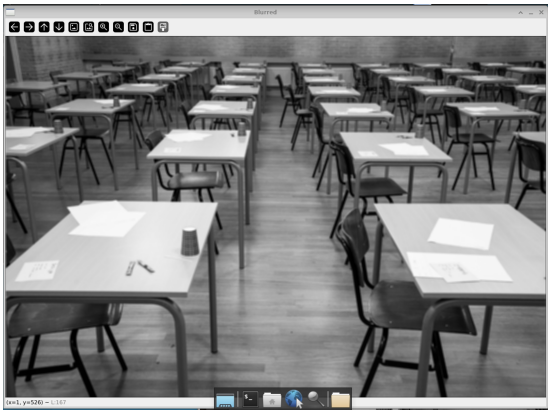
(a) Edges



(b) Sobel y



(c) Sobel x



(d) Blurred

Figure 6: Task 6 Results

Task 7

```
# Terminal A:
ros2 bag play path/to/your/lab3_2

Terminal B:
python3 task7.py
```

In Task 7 the edge detection from Task 6 was extended to work on a live video stream from a rosbag. The functions for convolution and Sobel filtering were integrated into a ROS2 subscriber, so that each incoming frame was processed directly. The edges were then calculated and displayed in real time, showing the original image, the gradients, and the combined edge result. This allowed the static approach from Task 6 to be applied continuously on streaming data.

Task 8

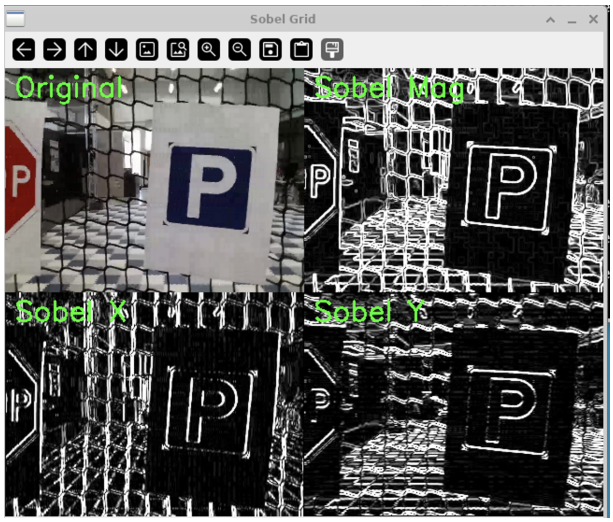


Figure 7: Task 7 Results

```
# Terminal A:
ros2 bag play path/to/your/lab3_2

Terminal B:
python3 task8.py
```

In Task 8 the edge detection approach was changed from the Sobel method used in Task 7 to the Canny algorithm. While Sobel mainly calculates gradients in the x- and y-directions, Canny includes additional steps such as noise reduction, gradient magnitude and direction, non-maximum suppression, and hysteresis thresholding. This makes the detected edges thinner, cleaner, and less noisy compared to Sobel. The implementation was again applied to frames from a rosbag in real time, and the results showed that Canny produces more precise edge maps, which are often preferred in practical applications.

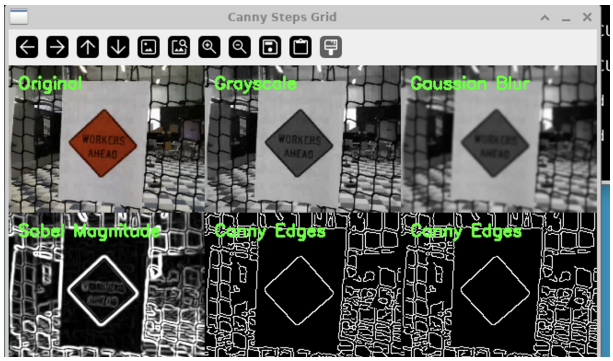


Figure 8: Task 8 Result

Task 9

```
# Terminal A:
ros2 bag play path/to/your/lab3_3
```

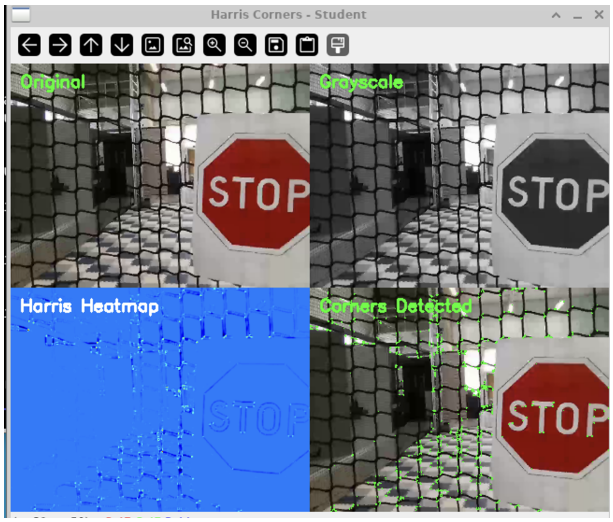


Figure 9: Task 9 Result

```
Terminal B:
python3 task9.py
```

In Task 9 Harris corner detection was applied to frames from the rosbag to highlight corner features in real time. The processed output was visualized with both a heatmap of the corner response and an overlay of detected corners on the original image, arranged together in a 2×2 grid for easier comparison.

The main parameters influence the quality of detection: block size defines how large the neighborhood is for calculating the response, with larger values giving smoother but fewer corners. ksize sets the size of the Sobel operator used for gradient calculation, where smaller values are sensitive to fine details and larger values smooth more strongly. The parameter k controls the sensitivity of the detector, balancing between edges and true corners. Finally, the threshold determines which responses are marked as corners, with higher thresholds showing only the strongest points and lower thresholds including weaker, possibly noisy detections.