Lab-02-Sensor Robotics

Students: Alexis GIBERT / Souha GHAZOUANI

1 Goals

In this practical session on Sensory Robotics, the primary objective is to utilize the Intel RealSense d435i camera system for segmenting RGB images based on their corresponding depth image-pairs and Python. The exercise involves capturing two sets of images: one with an empty scene and a person standing in the background, and the other with an empty scene but with a chair placed in the foreground. The challenge is to merge these RGB images based on their depth information in such a way that the chair obscures a portion of the standing person.

Participants are expected to develop a script that loads the saved .npy archives containing the images, displays the raw images, performs the fusion of RGB images based on depth information, and finally displays the fused result. The goal is to achieve a more sophisticated fusion solution than the simplest method provided, aiming for a reduction in errors and noise in the resultant image through standard image processing routines.

2 Several types of depth camera

The article "Beginner's guide to depth" outlines three main types of depth cameras along with their functionalities and applications.

- **Structured light and coded light depth cameras** utilize projected light patterns, usually in the infrared spectrum, onto the scene. These patterns deform or change when interacting with surfaces in the scene. By analyzing the distortion of the patterns captured by the camera, depth information is obtained. These cameras excel in indoor environments and are suitable for applications like gesture recognition or background segmentation.
- **Stereo depth cameras** (*like the Intel RealSense d435i camera used in this work-lab*) employ two sensors spaced a small distance apart, mimicking the human binocular vision. By comparing the images captured by these sensors, depth information is inferred from the disparities between corresponding points in the images. These cameras work well in various lighting conditions, including outdoor settings. They are suitable for applications requiring accurate depth perception and can be used in conjunction with multiple cameras without interference.
- **Time of Flight (ToF) and LiDAR cameras** rely on measuring the time it takes for emitted light to return to the sensor after reflecting off objects in the scene. This method utilizes the known speed of light to calculate distances. LiDAR, a type of ToF camera, employs laser light for depth calculation and is often used in applications such as terrain mapping or self-driving vehicles. However, ToF cameras can be susceptible to interference from other light sources and may perform less reliably in outdoor conditions.

3 Measurement Process

- Ensure that the Intel RealSense d435i camera system is properly set up and connected to the system.
- Run the modified_align_depth2color.py script, which initializes the camera and continuously displays the RGB and depth streams.
- Position the camera to capture the desired scenes: one image with a person and another with a chair.
- Press the appropriate key (e.g., 'c') to capture each pair of RGB and depth images simultaneously and save us as .npy arrays for further processing.
- Create a Python code to fuse the two depth pictures captured in the previous step.
- Display the fused image for analysis and further processing if needed.

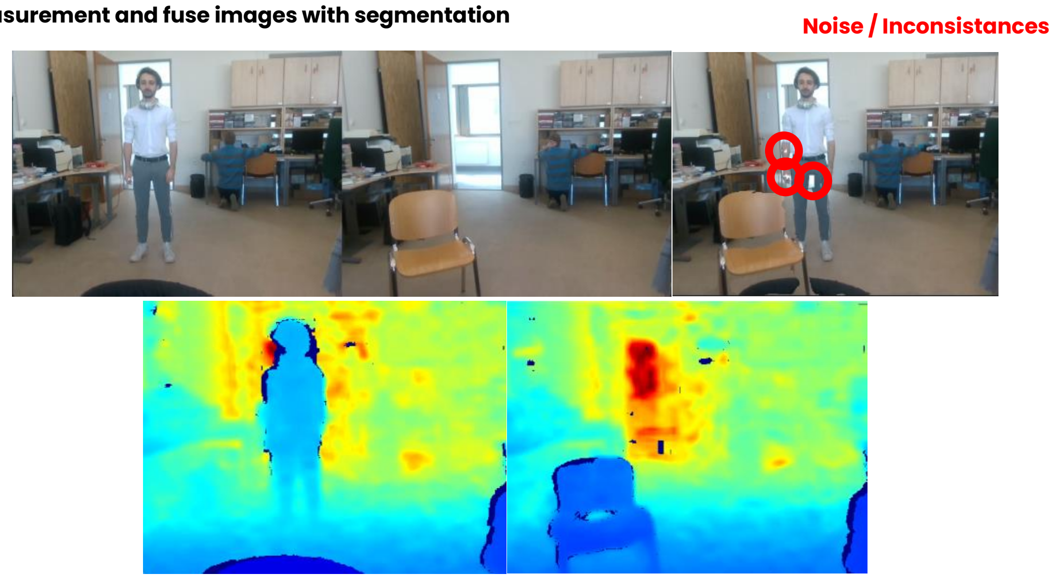
4 First results

In our initial measurements, we conducted two tests where we captured depth images and fused them with segmentation. However, both tests revealed significant issues. In the first measurement, we observed noise in the fused image, while in the second measurement, a portion of the image, specifically my leg, was erased.

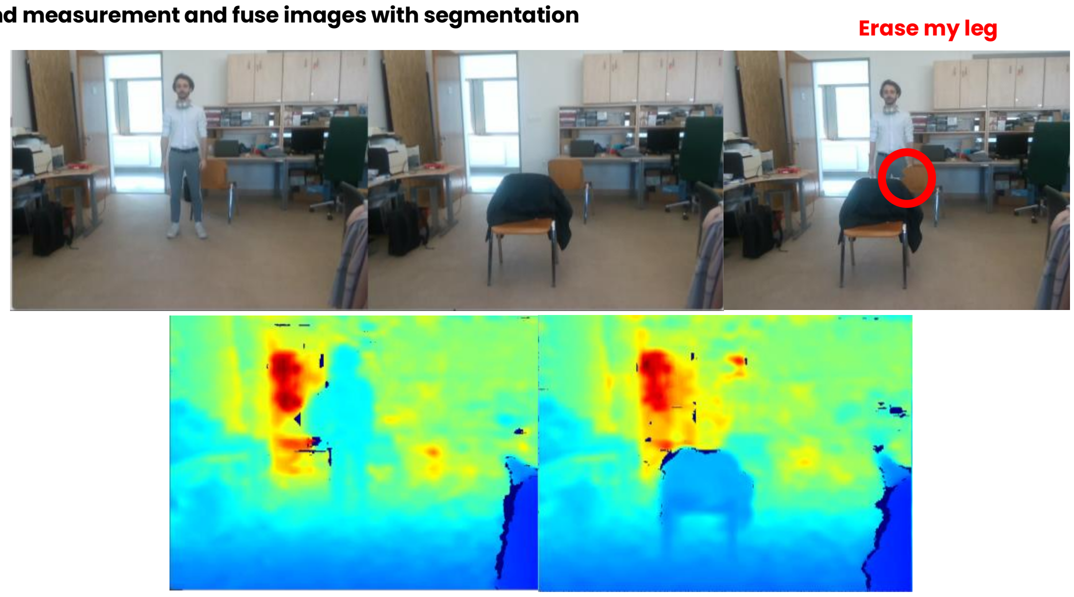
Upon analysis, we identified that the primary cause of these errors was the presence of dark blue points in the images, which represented holes in the depth data. These holes occurred due to inaccurate depth measurements by the sensor, resulting in missing or erroneous depth values.

To address this issue and improve the accuracy of our depth data, we decided to implement a filtering approach. Specifically, we applied a filtering technique to the depth data, filling in the holes by interpolating values from the surrounding areas. By utilizing the depth information from neighboring pixels, we effectively corrected the erroneous depth values and eliminated the holes present in the images.

First measurement and fuse images with segmentation



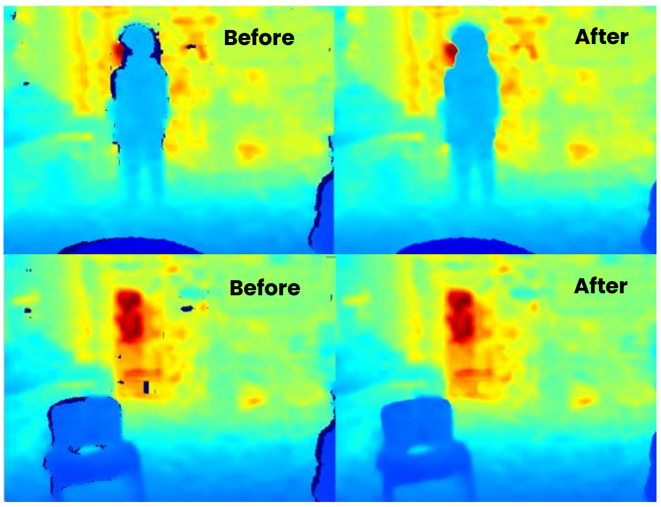
Second measurement and fuse images with segmentation

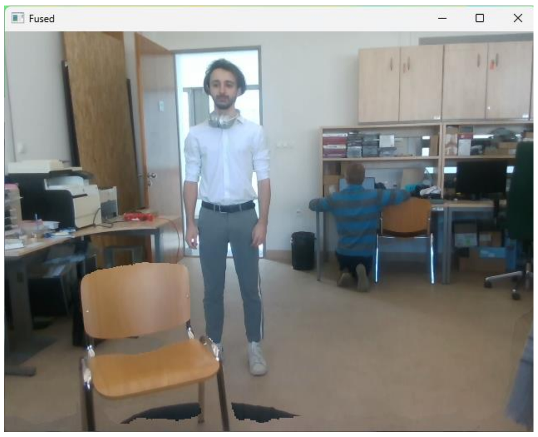


5 Final result

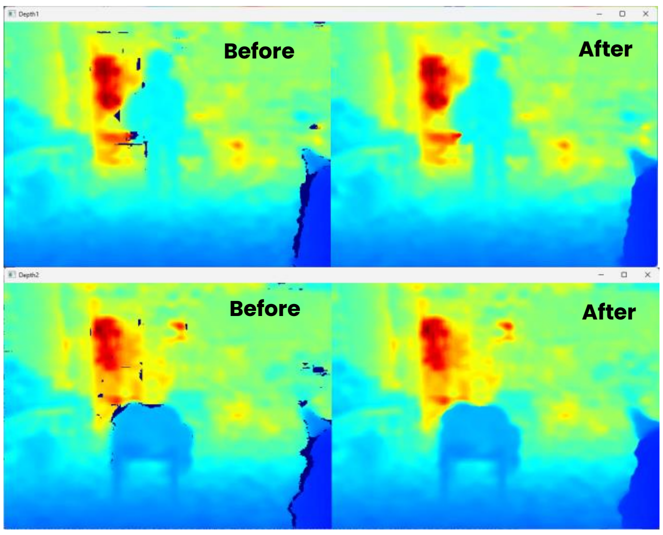
Following the filtering process, we observed significant improvements in the quality of the fused images. The holes that were previously present were successfully filled, resulting in a more complete and accurate representation of the scenes captured by the depth camera. Additionally, by adjusting the threshold parameters appropriately, we achieved optimal results, ensuring that the fused image was free from artifacts and inconsistencies.

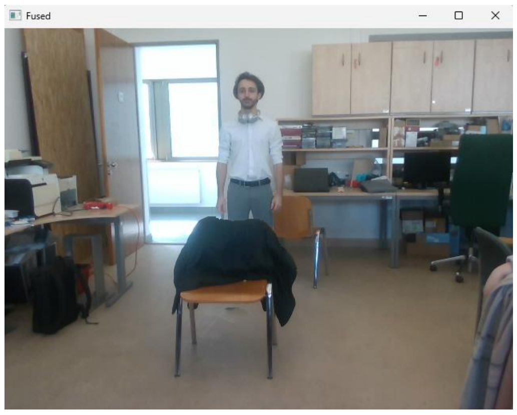
First measurement and fuse images with filtering and segmentation (depth_threshold constant = 20)





Second measurement and fuse images with filtering and segmentation (depth_threshold constant = 50)





In conclusion, the implementation of the filtering technique proved to be highly effective in improving the quality of the depth data and subsequently enhancing the fused images. With the successful correction of the depth data, we have achieved a more accurate and reliable representation of the captured scenes.

6 Final code

```
import numpy as np
import cv2

# Load data from .npy files
file1_RGB = np.load('20240320_093036.206357_color.npy')
file1_DEPTH = np.load('20240320_093036.206357_depth.npy')
file2_RGB = np.load('20240320_093104.161635_color.npy')
file2_DEPTH = np.load('20240320_093104.161635_depth.npy')

# Convert depth data to an appropriate format (CV_8U)
file1_DEPTH_uint8 = (file1_DEPTH / np.max(file1_DEPTH) * 255).astype(np.uint8)
file2_DEPTH_uint8 = (file2_DEPTH / np.max(file2_DEPTH) * 255).astype(np.uint8)

# Fill small holes in the depth image using neighborhood color
file1_DEPTH_filled = cv2.inpaint(file1_DEPTH_uint8, (file1_DEPTH_uint8 ==
0).astype(np.uint8), 3, cv2.INPAINT_TELEA)
file2_DEPTH_filled = cv2.inpaint(file2_DEPTH_uint8, (file2_DEPTH_uint8 ==
0).astype(np.uint8), 3, cv2.INPAINT_TELEA)

# Semi-automatic thresholding
depth_threshold = (np.mean(file1_DEPTH_filled) + np.mean(file2_DEPTH_filled)) / 2 - 20  #
Adjusted threshold

# Create a mask based on depth information
mask = file2_DEPTH_filled < depth_threshold
mask = np.repeat(mask[:, :, np.newaxis], 3, axis=2)

# Replace background parts of the image
fused_image = file1_RGB.copy()
fused_image[mask] = file2_RGB[mask]

# Convert depth data to grayscale images
file1_depth_image_gray_original = (file1_DEPTH / np.max(file1_DEPTH) *
255).astype(np.uint8)
file2_depth_image_gray_original = (file2_DEPTH / np.max(file2_DEPTH) *
255).astype(np.uint8)
file1_depth_image_gray = (file1_DEPTH_filled / np.max(file1_DEPTH_filled) *
255).astype(np.uint8)
file2_depth_image_gray = (file2_DEPTH_filled / np.max(file2_DEPTH_filled) *
255).astype(np.uint8)

# Apply false color to depth images
file1_depth_colored_original = cv2.applyColorMap(file1_depth_image_gray_original,
cv2.COLORMAP_JET)
file2_depth_colored_original = cv2.applyColorMap(file2_depth_image_gray_original,
cv2.COLORMAP_JET)
file1_depth_colored = cv2.applyColorMap(file1_depth_image_gray, cv2.COLORMAP_JET)
file2_depth_colored = cv2.applyColorMap(file2_depth_image_gray, cv2.COLORMAP_JET)

# Concatenate all images horizontally
depth1 = np.hstack((file1_depth_colored_original, file1_depth_colored))
depth2 = np.hstack((file2_depth_colored_original, file2_depth_colored))
color = np.hstack((file1_RGB, file2_RGB))

# Display the combined image
cv2.imshow('Depth1', depth1)
cv2.imshow('Depth2', depth2)
cv2.imshow('Color', color)
cv2.imshow('Fused', fused_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```