Project Work: Perception and Situation Understanding

Part: Prof. Dr.-Ing. Stache / N. Hessenthaler

Name: Jakob Kurz

Matrikel-Nr. / Student-ID: 210262

Workmate: Lukas Gerstlauer, 205293

I confirm that the work I handed in is my own work. All sources are cited.
I know that my work will be checked for plagiarism, and I accept that any occurrence of plagiarism and/or non-cited sources will lead to a grade of 5.0 (failed / nicht bestanden). **Group work is allowed in groups of 2**, but every student must hand in his/her own work and documentation. If you decided to work in a team, mention your workmate in the field above.

The due date for this work to upload to ILIAS is July 20, 2025, time: 23:55. Uploads after this due are not accepted.

0. File download – Additional_files.zip

The following three tasks require additional files. Therefore, a .zip file called Additional_files.zip is provided in Ilias. It contains the necessary additional files. The .zip file contains the following data.

- Images from stereo camera in *Stereo_images.zip*. The images from left and right camera are corresponding for each image number. The stereo camera images are rectified using a virtual camera. This allows the epipolar constraint to be assumed during the tasks. **The images are required for task 1 and 2.**

- The parameters of the virtual rectified camera are provided in the file *Camera.txt*. The parameters in the section [INTERNAL] describe the intrinsic parameters of both rectified cameras. The parameters in the section [EXTERNAL] describe the extrinsic parameters of the left rectified camera, which is the reference camera. **The positions of the camera describe the origin of the left rectified camera in the world coordinate system already!** The angles must be processed as described in task 1. **The camera.txt file is required for task 1.**

- Lidar point cloud data of two scenes of the Kitti dataset given in **2011_09_26_drive_0001.** The corresponding .zip file is called *Lidar_data.zip*. For each scene, the following data is provided:
    a. .png for scene understanding
    b. .pcd containing modified / preprocessed lidar point cloud
    c. .txt containing odometry data (location)

  The files with the same number belong together and refer to one scene each. **The lidar point cloud data is required for task 3.**

1. Point projection and coordinate transformation

Open the *Stereo_images.zip* which you have downloaded in the previous task. Search for the image *image0110_c0.pgm*. The image should be the same image as shown below.

Find the pixel position of the bottom right-hand corner of the traffic sign. The corner is marked with a red circle in the image above.

Enter the pixel position of the marked corner of image *image0110_c0.pgm*:

$u_l =$ 250 Pixel, („X-coordinate in the image")

$v_l =$ 250 Pixel, („Y-coordinate in the image")

Do the same for the corresponding right image *image0110_c1.pgm*:

$u_r =$ 238 Pixel

$v_r =$ 250 Pixel

Calculate the pixel-accurate disparity from the values above:

$d =$ 12 Pixel

Where is this point of the traffic sign located in the world coordinate frame? Perform a projection of this point to the camera coordinate frame using the intrinsic parameters as provided in the section [INTERNAL] of the file *Camera.txt*. Then, perform a transformation of the point in the world coordinate frame. This can be done by using the extrinsic parameters as provided in the section [EXTERNAL] of the file *Camera.txt*. Like all examples from the lecture, the world coordinate frame is defined according to DIN 70000! Thus, the x-axis points to the front, the y-axis points to the left and the z-axis points to the top.

Note 1: The positions of the camera describe the origin of the left rectified camera **in the world coordinate system already**!

Note 2: The angles of the camera describe the rotation **from world to camera coordinate** frame.

Note 3: To create a rotation matrix out of tilt, yaw and roll angle, assume the following:

- Rotations are around fixed rotation axes

- Order of rotation is defined as: First apply yaw Ψ, then tilt Θ, and finally the roll angle Φ

→ This results in a rotation matrix of:

$$M_{GNR} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi & \sin\Phi \\ 0 & -\sin\Phi & \cos\Phi \end{pmatrix} \begin{pmatrix} \cos\Theta & 0 & -\sin\Theta \\ 0 & 1 & 0 \\ \sin\Theta & 0 & \cos\Theta \end{pmatrix} \begin{pmatrix} \cos\Psi & \sin\Psi & 0 \\ -\sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$M_{GNR} = \begin{pmatrix} \cos\Theta\cos\Psi & \cos\Theta\sin\Psi & -\sin\Theta \\ \sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi & \sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi & \sin\Phi\cos\Theta \\ \cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi & \cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi & \cos\Phi\cos\Theta \end{pmatrix}$$

(Source: https://de.wikipedia.org/wiki/Eulersche_Winkel#Standard-x-Konvention_.28z.2C_x.E2.80.B2.2C_z.E2.80.B3.29, 19.12.2016)

Note 4: Consider the coordinate system definition of DIN70000!

Provide the position of the bottom right-hand corner of the traffic sign in the world coordinate frame.

${}^{W}P_x = 22.974\ \mathrm{m}$ (X-coordinate in world coordinate frame)

${}^{W}P_y = 1.119\ \mathrm{m}$ (Y-coordinate in world coordinate frame)

${}^{W}P_z = 2.380\ \mathrm{m}$ (Z-coordinate in world coordinate frame)

Write down all processing steps in this documentation. You shall enter equations and paste Python printout below this line of the document. Explain the steps the lead to the result printed above. **Additionally, validate your calculations.** You can validate the result by evaluating objects of known size / known relations that are visible in the image.

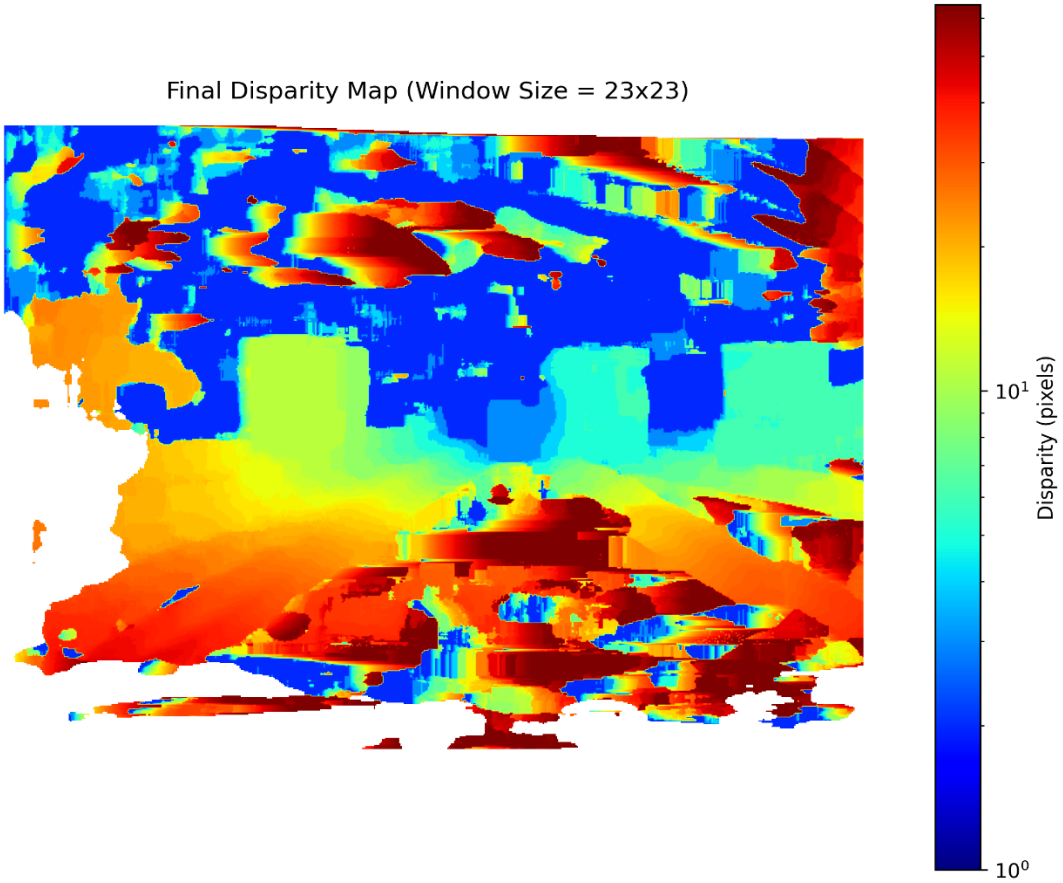
Hand in all Python files that were used to process this task. The code must be fully executable. Make sure the code style is like the style used in the demo examples.

Hint: A bonus point will be granted for extensive validations.

2. Stereo Disparity Map

Compute a stereo disparity map for the *images image0110_c0.pgm* and *image0110_c1.pgm* that are provided in Stereo_images.zip. Use normalized cross correlation as similarity metric. Your implementation should reflect the algorithms as discussed in the lecture. **Do not use** available implementations of Python libraries to create your disparity map (e.g. OpenCv `stereo = cv.StereoBM_create()`. Instead, implement your own algorithm to create the disparity map. It is allowed to use supporting libraries like Numpy or existing implementations of the normalized cross correlation. Try to find an optimal window size.

Paste the disparity image here. Use the colormap "jet" for coloring the disparity image. (https://matplotlib.org/stable/users/explain/colors/colormaps.html) Please enable the logarithmic scaling for the colormap.



I used the window-size _23_ x _23_ Pixel.

Hand in all Python files that were used to process this task. The code must be fully executable. Make sure the code style is like the style used in the demo examples.

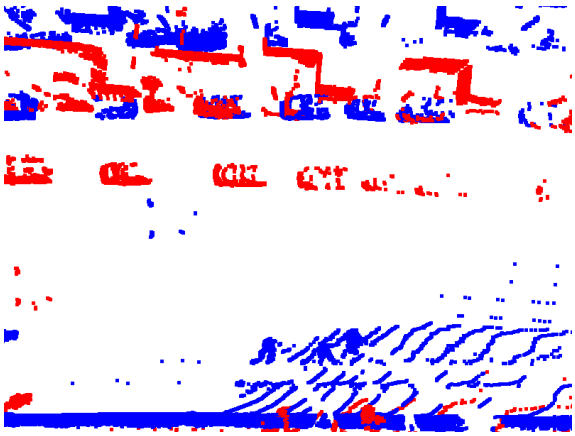
Hint: A bonus point will be granted for work beyond the scope discussed in the lecture (e.g. runtime optimization, additional filters, …) or extensive validations.

3. Lidar Data and ICP

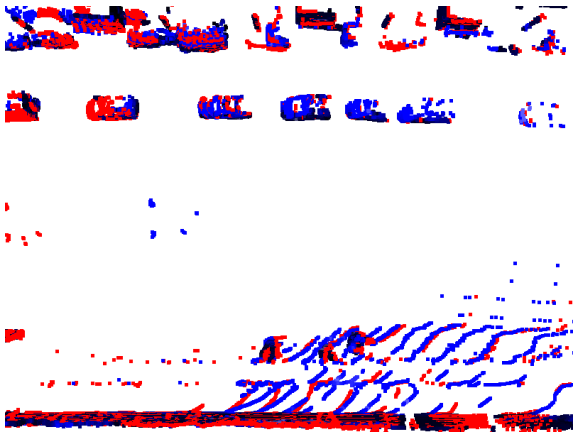
Perform the following steps to match lidar point cloud data via the iterative closest point algorithm as discussed in the lecture.

1. Extract Lidar_data.zip from the overall .zip that was downloaded in step 0.
2. The folder contains two scenes of the Kitti dataset given in **2011_09_26_drive_0001.**
3. For each scene, the following data is provided:
    a. .png for scene understanding
    b. .pcd containing modified / preprocessed lidar point cloud
    c. .txt containing odometry data (location) → see following page for details
4. Develop a python program that can read in the provided point clouds of both scenes given in 0000000020.pcd and 0000000030.pcd. **The format of the given point clouds is compatible already with the Python library open3D.**
5. With the given data, think of a way to provide an initial scene transformation. Rotation can be omitted.
6. Find a good distance threshold for the ICP process to perform the initial point matching.
7. Plot the point clouds after the initialization step. This should provide a good overview of the quality of the initialization.
8. Perform the ICP algorithm to iteratively refine the point cloud matching. You are allowed to use the implementations provided in the open3D library. Find sufficient parameter values for the ICP.
9. Plot the point clouds after the ICP process.

Insert the plots of the point clouds after the initialization and after the ICP process in this document (**top view, same angle for good comparability**). Explain, how you determined the initialization values for the ICP. Justify the chosen distance threshold. Paste the final transformation that was determined by the ICP into this document and interpret the results (Does this match to the expected trajectory as seen in the images?)



Initial Alignment



Alignment after ICP

Final Transformation:

$$\begin{bmatrix} 0{,}999 & -0{,}026 & 0{,}000294 & -11{,}759 \\ 0{,}0266 & 0{,}999 & 0{,}0024592 & -0{,}203 \\ -0{,}000360 & -0{,}00245 & 0{,}999 & -0{,}0985 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Hand in all Python files that were used to process this task. The code must be fully executable. Make sure the code style is like the style used in the demo examples.

GPS/IMU 3D localization unit

The GPS/IMU information is given in a single small text file which is written for each synchronized frame. Each text file contains 30 values which are:

- lat: latitude of the oxts-unit (deg)
- lon: longitude of the oxts-unit (deg)
- alt: altitude of the oxts-unit (m)
- roll: roll angle (rad), 0 = level, positive = left side up (-pi..pi)
- pitch: pitch angle (rad), 0 = level, positive = front down (-pi/2..pi/2)
- yaw: heading (rad), 0 = east, positive = counter clockwise (-pi..pi)
- vn: velocity towards north (m/s)
- ve: velocity towards east (m/s)
- vf: forward velocity, i.e. parallel to earth-surface (m/s)
- vl: leftward velocity, i.e. parallel to earth-surface (m/s)
- vu: upward velocity, i.e. perpendicular to earth-surface (m/s)
- ax: acceleration in x, i.e. in direction of vehicle front (m/s^2)
- ay: acceleration in y, i.e. in direction of vehicle left (m/s^2)
- az: acceleration in z, i.e. in direction of vehicle top (m/s^2)
- af: forward acceleration (m/s^2)
- al: leftward acceleration (m/s^2)
- au: upward acceleration (m/s^2)
- wx: angular rate around x (rad/s)
- wy: angular rate around y (rad/s)
- wz: angular rate around z (rad/s)
- wf: angular rate around forward axis (rad/s)
- wl: angular rate around leftward axis (rad/s)
- wu: angular rate around upward axis (rad/s)

- posacc: velocity accuracy (north/east in m)
- velacc: velocity accuracy (north/east in m/s)
- navstat: navigation status
- numsats: number of satellites tracked by primary GPS receiver
- posmode: position mode of primary GPS receiver
- velmode: velocity mode of primary GPS receiver
- orimode: orientation mode of primary GPS receiver