# Project Work: Perception and Situation Understanding

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

Name: Lukas Gerstlauer

Matrikel-Nr. / Student-ID: 205293

Workmate: Jakob Kurz

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:
  1. .png for scene understanding
  2. .pcd containing modified / preprocessed lidar point cloud
  3. .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*:

Pixel, („X-coordinate in the image”)

Pixel, („Y-coordinate in the image”)

Do the same for the corresponding right image *image0110\_c1.pgm*:

Pixel

250Pixel

Calculate the pixel-accurate disparity from the values above:

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**!

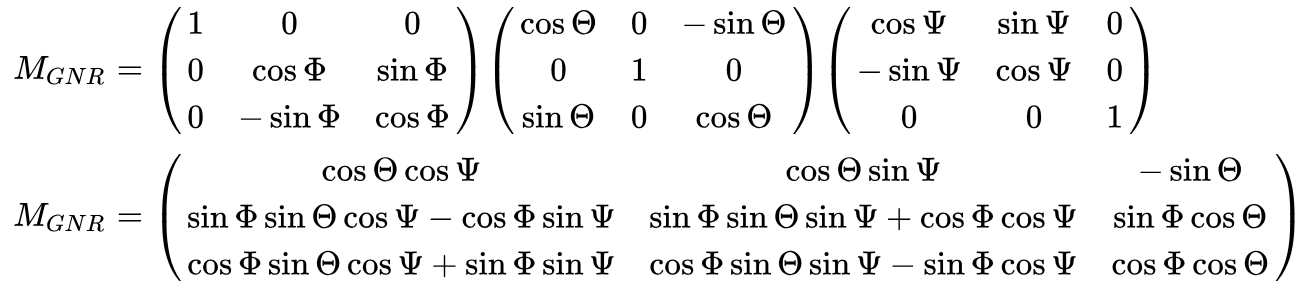
**Note 2:** The angles of the camera describe the rotation **from world to camera coordinate** frame.

**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 , then tilt , and finally the roll angle

🡪 This results in a rotation matrix of:

  
(Source: [https://de.wikipedia.org/wiki/Eulersche\_Winkel#Standard-x-Konvention\_.28z.2C\_x.E2.80.B2.2C\_z.E2.80.B3.29](https://de.wikipedia.org/wiki/Eulersche_Winkel), 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.

(X-coordinate in world coordinate frame)

(Y-coordinate in world coordinate frame)

(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.

Replace by disparity image

I used the window-size \_\_\_\_\_\_\_\_\_\_x\_\_\_\_\_\_\_\_\_ 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.