Inhalt

[Architecture 2](#_Toc503357687)

[Functionality of the Modules 3](#_Toc503357688)

[Edge Detection 3](#_Toc503357689)

[Concatenation 3](#_Toc503357690)

[Coordinate Transformation 3](#_Toc503357691)

[Vector Fitting 4](#_Toc503357692)

[Middling 4](#_Toc503357693)

[Calculation of the Ego-Lane-Points 4](#_Toc503357694)

[Goals and Improvements 4](#_Toc503357695)

# Architecture



# Functionality of the Modules

The following sub sections provide a sort of pseudo code to better understand how each sub module inside the lane detection-node is functioning.

## Edge Detection

* Select the region of interest (ROI)
* Apply a color mask, so we only find the white stripes on inside the picture
  + We now have only two colors inside the picture: black and white
* Calculate the gradient map of the picture via the “Scharr”-function
  + This is similar to the Sobel-filter
* Pass the gradient maps for the x- and y-gradients on to the concatenation module
  + Currently only the x-gradient is actually used to find the lane inside the picture, but the y-gradient map is still passed on

## Concatenation

* Collect the points based upon their gradient
  + Put all points with a x-gradient greater than the gradient-threshold and positioned on the right side of the image into a vector/array for the for the right lane marking
  + Put all points with a x-gradient smaller than the negative gradient-threshold and positioned on the left side of the image into a vector/array for the for the left lane marking
* Sort out all points on the left and right side, that are not considered near enough to be part of the lane markings ( currently based upon pixel distance )
  + The Points at this point are sorted descending based upon their y-value  
    (this is important for the fitting algorithm)
* Pass on the left and right lane markings as vectors/arrays of points

## Coordinate Transformation

* Transform the points into real world coordinates in SI-Units based upon the formulas seen in “**Herleitung der Realdistanz vom Bildpixel**”
* Pass the transformed points on to Vector fitting

## Vector Fitting

* Fit a vector function to both the left and right lane marking
  + You can find more Details on this in: “**Vector Function Fitting**”

### Middling

* Simply take all the values of the left and right lane marking and find the average for each of them
* Remove the artifact at the beginning of the function

## Calculation of the Ego-Lane-Points

* Depending on the number of points that need to be calculated, split the length of the vector function into n+1 sections
* At the end of each section, calculate the tangent, curvature, and x-y-coordinates
  + Put each of these values into a separate array
* Publish these four arrays via ROS
  + This message then will be picked up over the message-handler of Mav-Link and will be send to the STM-Board

# Goals and Improvements

Under the current performance, the Improvements are planned for future releases:

* Better filtering of interference points, such as:
  + Similar Gradients to the lane (such as the wall)
  + light variability
  + reflective Surfaces
* Adding a Tracker similar in concept or identical to a Kalman-Filter
* Tracking of the ego lane based upon the past detected lane and the ego-motion