Technische Universität Dresden

Fakultät Elektrotechnik und Informationstechnik

Institut für Regelungs- und Steuerungstheorie

Studienarbeit

Verbesserung der Spurerkennung und -verfolgung autonomer Modellfahrzeuge

vorgelegt von:	James Vero Asghar
geboren am:	3. Dezember 1997 in Austin, Texas, Vereignete Staaten

Betreuer:	Dr.-Ing. Carsten Knoll
	M.Sc. Paul Auerbach
Verantwortlicher Hochschullehrer:	Prof. Dr.-Ing. habil. Dipl.-Math. K. Röbenack
Tag der Einreichung:	2. Februar 2222

Bitte ersetzen Sie diese Seite vor dem Binden mit der Aufgabenstellung.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tage an der Fakultät Elektrotechnik und Informationstechnik eingereichte Studienarbeit zum Thema

Verbesserung der Spurerkennung und -verfolgung autonomer Modellfahrzeuge

selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Schriften entnommen sind, wurden als solche kenntlich gemacht.

Dresden, 2. Februar 2222 James Vero Asghar

Kurzfassung

An dieser Stelle fügen Sie bitte eine deutsche Kurzfassung ein.

Abstract

At the Connected Robotics Lab (CoRoLa) research group at the Barkhausen Institut, a demonstration using remote controlled 2 axis vehicles was developed. The demonstration is currently in use in order to model communications between vehicles similar to an Internet of Things (IoT) network.

In order to better model the complexities of a real world system a nonlinear control algorithm was introduced to the system in order to more accurately control the vehicles. The Stanley Controller is a nonlinear controller developed for use with the lateral control of 2 axis vehicles in mind. As input to the controller, a fisheye camera was used to create images of road. These images were then digitally processed to find the yellow line for the vehicles to follow.

Inhaltsverzeichnis

Kapitel 1

Einführung

Kapitel 2

Regelalgorithmus

2.1 Stanley-Regler

The Stanley controller is a non-linear control algorithm that was developed in 2005 by Stanford University with the purpose of controlling a two-axle vehicle using only the heading and the cross track error from the path to be followed. The algorithm has been proven to be asymptotically globally stable for the kinematic model of a two-axle vehicle. However, it should be noted that the model used in this thesis accounts for the dynamics of the steering angle rather than being based solely on the kinematic model of the vehicle.

The Stanley controller is a path following controller rather than a trajectory following algorithm. As a lateral controller, it is designed to keep the vehicle on the path but has no impact on the speed at which the vehicle travels along the path. This approach allows for a flexible choice of the vehicle speed, which can be selected as required for a given application, subject to dynamic effects.

The Stanley controller is represented mathematically by the following equation:

$$u = \theta - \theta_d + \arctan\left(\frac{ke_{fa}}{v}\right), \tag{2.1}$$

where u represents the controller output, θ is the current heading of the vehicle, θ_d is the heading of the path, k is a scaling factor, v is the velocity of the vehicle, and e_{fa} is the cross track error from the midpoint of the vehicle's front axle to the path.

The Stanley controller consists of two components: a component that handles the offset of the vehicle from the path and a component that handles the derivative of the offset (difference in heading). The first component is represented by $\arctan(\frac{ke_{fa}}{v})$ and drives the steering angle to turn more towards the path as the car gets farther away from it. The second component, represented by $\theta_d - \theta$, controls the steering angle to stay parallel with the path to be followed. Together, these two components work to steer the vehicle onto the desired path.

2.2 Simulation

In order to investigate the behavior of the Stanley controller on the remote controlled vehicle, a simulation of the control loop was used. The simulation has 3 major components: first is the path that the vehicle must follow, second is the Stanley controller and third is the vehicle model. A differential equation solver is used in order to analyze the time series of the 2 axle vehicle.

The path that the vehicle must follow is a virtual version of the real world track that the vehicle will use. A visual representation of the path is FIGURE. In the simulation, the path is represented with the following continuous static function: —, with the input t and the output (x, y, h). This function is then discretized for any chosen amount of t values, in order to create an array of (x, y, theta) values. This array is then fed into the differential equation solver that encapsulates the Stanley controller and the vehicle model.

The vehicle model used is the rear axle vehicle model. This model is represented by the following nonlinear state space representation:

$$\dot{x} = v\cos(\theta) \tag{2.2}$$

$$\dot{y} = v\sin(\theta) \tag{2.3}$$

$$\dot{\theta} = \frac{v}{l}\tan(\varphi) \tag{2.4}$$

$$\dot{\varphi} = \frac{-1}{T}(\varphi - \delta)\,. \tag{2.5}$$

The state variables of the model are x_H, y_H, ϕ und θ, where x_H und y_H are the XY-coordinates of the midpoint of the rear axle and θ is the heading of the vehicle. ϕ is the steering angle of the vehicle. The input variables of the model are v und ϕ_d, where v is the velocity of the vehicle and φ is the output of the Stanley controller. The steering angle φ is represented by a first order linear differential equation with the configurable time constant T The behavior of the Stanley controller when subjected to dynamic effects on the steering angle, is of great importance, as the global asymptotic stability of the controller was only proven on the kinematic model, with a springable steering angle.

As the Stanley controller requires the midpoint of the front axle for the cross track error, it must be calculated from the rear axle. The midpoint of the front axle is calculated from the state variables through the following static function:

$$\underline{p}_H := \begin{bmatrix} x_H & y_H \end{bmatrix}^T \tag{2.6}$$

$$\underline{p}_V := \begin{bmatrix} x_V & y_V \end{bmatrix}^T \tag{2.7}$$

$$\underline{p}_V = \underline{p}_H + l\begin{bmatrix} \cos(\theta + \pi/2) \\ \sin(\theta + \pi/2), \end{bmatrix} \tag{2.8}$$

where xf and yf are the x/y coordinates of this midpoint.

As seen before, the Stanley controller is composed of multiple parts, which must be calculated. The heading is a state variable, therefore it is always available. For the cross track error and the heading path, the correct path point must be chosen. The Stanley controller uses the closest path point to determine the required heading and current cross track error. For the simulation this is determined by finding the point with the smallest distance from the front axle midpoint. The algorithm is represented as follows: —. Through this found point, the heading path is determined. The cross track error is then determined through the following equation: —. This equation represents the dot product of the vector perpendicular to the heading vector and the vector from the front axle to the path point. FIGURE is a visual representation of these two vectors. The heading path and cross track error are then fed into the Stanley controller, whereby the output of the controller is fed into the vehicle model.

As the Stanley controller only outputs a steering angle, the velocity is assumed to be constant. A high level diagram of the control loop is represented in FIGURE.

The Stanley controller works with continuous time signals, which is different from what is used on the vehicle. The vehicle captures images at a specific sampling frequency, which are then processed by the pipeline before being fed into the controller. In order to simulate this behavior, a cache was built into the simulator, where the heading path and cross track error are saved. The Stanley controller is then fed these saved values for a chosen amount of time. By caching these values, the behavior of a zero-order hold is simulated. The output of this zero-order hold is then fed into the Stanley controller. This allows for the behavior of the vehicle to be investigated for different sampling frequencies, which allows for a tolerance to be set for the processing speed of the pipeline. A comparison of the simulation at various sampling frequencies is shown in FIGURE.

As can be seen in FIGURE, the Stanley controller is indeed affected by the sampling frequency, however above X Hz the behavior of the vehicle is not noticeably different.

The simulation is then executed for a chosen amount of time.

$$\underline{x} := \begin{bmatrix} x_H & y_H & \theta \end{bmatrix}^T \tag{2.9}$$

$$\underline{\nu} := \begin{bmatrix} -\cos(\theta + \pi/2) \\ -\sin(\theta + \pi/2) \end{bmatrix} \tag{2.10}$$

$$\underline{p}_P := \begin{bmatrix} x_P & y_P \end{bmatrix}^T \tag{2.11}$$

$$e_V := \underline{\nu} \cdot \min_i |\underline{p}_V - \underline{p}_{Pi}|? \tag{2.12}$$

Kapitel 3

Bildverarbeitung

3.1 Kamera-Kalibrierung

The start of the image processing pipeline is the camera calibration. In this project, a fisheye camera was chosen as opposed to a rectilinear camera. The benefit of a fisheye camera arises from its wider angle of view in comparison to a rectilinear camera. However, a fisheye camera distorts(?) an image differently from a rectilinear camera. A rectilinear camera preserves straight lines, when no distortion is present, as opposed to a fisheye camera, which will always curve straight lines. The curvature of these straight lines is dependent on their radial distance from the center of the image. An example is in FIGURE.

Camera calibration is used in order to approximate the extrinsic and intrinsic parameters of a camera (MATLAB). A calibrated camera allows for 3-D information to be recovered from a 2-D image. The intrinsic parameters of a camera are often represented by the following 3x3 matrix:

$$K = matrix$$

, where fx, fy, cx, and cy MEAN SOMETHING. The extrinsic parameters are often represented by the following row vector:

$$vector$$

, where R is the rotation of the camera with respect to a world frame and t is the translation from the same world frame. The row vector-matrix multiplication of the extrinsics vector and the intrinsics matrix result in the camera matrix P.

A camera is calibrated by using a collection of photos with known straight lines. Commonly, a series of checkerboard images with known dimensions are used. Photos of the checkerboard are then captured at varying angles and locations in the scene. This series of images is then fed to the camera calibration algorithm, which will first

determine the locations of the checkerboard squares and the lines connecting them. Afterwards, using the model of a fisheye camera, the algorithm compensates for the distortion of the fisheye camera. For example, as seen in FIGURE, the curved lines of the checkerboard are made straight again after calibration. The number of needed images is dependent on the specific camera, however a large collection of photos will lead to a more accurate parameter approximation. (OPENCV Citation) Using images with a higher resolution will also lead to a more accurate parameters. (OPENCV/BLOG Citation) However, the approximated parameters are only accurate for the calibration of images taken at the same resolution as the images used for the calibration. In order to use the camera matrix for images with a lower resolution, the camera must be scaled with the following formula: (EQUATION). As the camera matrix is an example of an affine transformation, the value at $C_{3,3}$ must be reset to 1. The resulting camera matrix will work for smaller resolutions matching the aspect ratio of the images used for calibration, but it has been noticed that when the difference in resolution is too large, distortion is added back into the image. In FIGURE, the original image is on the left, the middle image is calibrated using images at 900p and the right image is calibrated using images at 240p. As can be seen, the image on the right straightens the ruler, while the middle image leaves it curved.

One downside to camera calibration is that any calibrated image has a lower resolution than the original image. This is a consequence of the calibration process, as the process will distort a subset of the image, particularly involving the pixels around the corners of the image. Therefore the calibration process will remove these pixels from the resulting image. In order to recreate the image in its original resolution, an interpolator is used. However, the interpolator will return a blurrier image than the original. In order to compensate for this, it is recommended to calibrate a camera with high resolution images and then capture images at that resolution. Then, instead of using the interpolator, shrink the images down to a resolution that is necessary for the application. This results in a more accurate image without any blurriness. Unfortunately, this process is computationally heavy and with the processor used on the vehicle, it was decided to only use the images from the interpolator in order to increase the processing speed.

Another downside from the camera calibration as that the midpoint of the camera shifts. An example of this can be seen in FIGURE. $C_{3,1}$ and $C_{3,2}$ represent the image center, with $C_{3,1}$ being the x coordinate. By manually changing this value, the midpoint of the image can be shifted back to its original location.

3.2 Color Thresholding

After the camera calibration comes the color thresholding stage of the pipeline. This stage removes all pixels from the image that are not yellow with a high saturation and brightness.

At first, the image is filtered of all pixels without a high red component, as bright yellow in the red, green and blue (RGB) color space has a high red component. Afterwards, the image is converted into the Hue, Lightness and Saturation (HLS) color space.

Under the RGB color space, yellow is represented as a combination of the red and green channels with the blue channel set to 0. Pure yellow is also defined as having both the red and green channels equal to one another, therefore allowing only 1 degree of freedom to tune the pipeline. Having only 1 degree of freedom leads to issues with tuning, as this reduces the ability for the pipeline to account for disturbances, such as differing lighting conditions or reflective surfaces. Under the HLS color space, the hue channel selects the color, the lightness channel corresponds to the amount of white or black in the color and the saturation channel is a measure of the purity of a hue. (MS ANNO) Once the hue is selected, here yellow, the lightness and saturation channels are then used to select the specific shade of yellow. Using these 2 channels allows for a more robust detection of color.

In order to detect the yellow lane, the pipeline will filter out colors outside of the yellow hue range, with a small tolerance, and then truncate the lower part of the lightness and saturation channels. As a result of this, only relatively pure yellow is left in the image. The values used in the pipeline for yellow is represented by the following range, TABLE.

An example of the output from this stage of the pipeline is FIGURE.

3.3 Perspective Transformation

The third part of the image processing pipeline is the perspective transformation. Whenever an image is captured with a camera mounted to the vehicle, the lane line will be trapezoidal as opposed to straight and lane seems to end in a point at the center of the image. This perspective requires that all calculations regarding the lane line must compensate for the decreasing width of the line. Therefore, to remove this requirement, a perspective transformation is employed. Perspective transformation is the process by which a subset of an image is sheared and made to fit the entirety of a new image. An example is in FIGURE. For this project, the shape of the subset of the image is selected to be a trapezoid, shown in FIGURE. Using this perspective transform, the trapezoidal shape of the lane line is corrected into one that is straight, shown in FIGURE.

A consequence of this new perspective, is that the resulting image is similar to a 2-D plane of the track. The perspective transformation simplifies further image processing as well, as all objects outside of the trapezoid are cropped, leaving only the track. However, as can be seen in FIGURE, the perspective transform introduces extra noise into the image. The shearing caused by the perspective transform causes pixels from the original image to be stretched, thereby polluting the image with noise.

Therefore, this is the third stage in the pipeline. As already discussed, in the second stage of the pipeline, the threshold stage removes the majority of unnecessary information from the image, which reduces the amount of noise caused by the perspective transform.

3.4 Histogram

After the perspective transformation stage, a histogram of the pixel density is created. During this step, the start point for the sliding window method is found. At every column of the image, the number of white pixels are counted. The numbers are then placed into a list. The assumption is then that the columns containing the largest numbers contain the lane information. The index with the number is then selected as the start point for the sliding window method. This pipeline stage reduces computation time by filtering out unnecessary parts of the image not containing lane information. A visual representation of the histogram can be seen in FIGURE.

3.5 Gleit-Fenster-Verfahren

In the fifth stage of the pipeline, the heading and offset of the lane are calculated through the sliding window method.

First, a chosen number of rectangles (windows) are created. The windows are aligned in a column and equally sized. Their height is chosen so that the column spans the entire height of the image, while their width is chosen to be a factor of the image width. The number of windows is chosen so that the column entirely or almost entirely contains the white pixels representing the lane, as can be seen in FIGURE. If the number of windows is too high, there is a chance that the method will falsely determine white pixels deep into the image as a part of the lane.

Second, the x-coordinate of the midpoint of the first window is set to the peak of the histogram from the previous stage. Then, the number of white pixels are then counted inside the window. If the number is above a chosen threshold, the pixels are added to an array and the mean of the x-coordinates of the contained pixels is calculated. The x-coordinate of the next window is set to this calculated mean while being placed on top of the first window. This process is then repeated for all remaining windows. When no pixels are contained in a window, the window is ignored and the following window is placed on top of it. In the image coordinate system, the top left corner of the image is $(0, 0)$, with the positive x-axis increasing to the right, while the positive y-axis increases moving down the image. Therefore, the bottom edge of the image would be any coordinate with a y component equal to the height of the image. Seen from the engineer's perspective, the windows would be stacked from lowest to highest, however

in the image coordinate space it is inverted. An example can be seen in FIGURE.

The pixels not located within each window are then filtered out of the image. The image before and after the filtering process can be seen in FIGURE. Using the the least squares method, a polynomial is approximated through remaining pixels.

3.6 Berechnung von Ausrichtung und Abstand

In the final stage of the pipeline, the heading and offset from the lane line are calculated.

From the fitted polynomial calculated in the previous stage, an analytical heading is found at a chosen point along the path with the following equation:

$$\theta = \arctan(f'(y))$$

. The inverse tangent of the derivative of the polynomial returns the heading at a chosen point. For the Stanley controller, only the difference between the path heading and vehicle heading is needed as there is no global frame for the vehicle, the heading of the vehicle is chosen to be zero degrees at all times. PICTURE WITH VISUAL REPRSENTATION

The classic Stanley controller, as described in chapter 2, uses the heading calculated from the nearest path point. However, as a side effect of the polynomial fitting, the heading calculated at the bottom of the image is incorrect. As can be seen in FIGURE, the sliding window method can on average not detect the lane line close to the bottom of the frame. Due to this, the fitted polynomial incorrectly approximates the heading. Therefore, the heading is calculated at a point located within the detected lane line instead. Moreover, this adds a degree of freedom to the controller. The consequences of this are discussed in chapter 4.

The offset from the lane line is calculated from the fitted polynomial with the following function:

$$e_{fa} = (\frac{w}{2} - f(h)) \cdot x_{mpp},$$

where w is the width of the image, $f(h)$ is the output of the fitted polynomial at the bottom of the image, x_{mpp} is a scaling constant to translate pixels into meters and e_{fa} is the offset from the vehicle to the lane. All together, the difference of the x-component of the midpoint and the x-component of the polynomial point located on the bottom edge of the image is translated into meters.

3.7 Ausreißer

A side effect of the sliding window method is a phenomenom termed öutliers". Outliers are any frame where the lane line is incorrectely determined. In FIGURE, a sequence of three frames with an outlier are observed with the left frame being the start. In the first frame, the lane line is correctly determined. In the second frame, the lane line is incorrectly determined. In the last frame, the lane line is once again correctly determined. The incorrect selection of HSV values is the largest contributor to the occurence of outliers. When the HSV values are not restrictive enough, colors besides yellow are not filtered out of the image. Which leads to incorrect determination of the lane line. Outliers are most common on the curves of the track, due a disadvantage of the pipeline implementation. As discussed previously, the first window of the sliding window method is selected at the x-position of the peak of the pixel histogram. Along the straight part of the track, the lane line is the densest object in the image. Therefore the peak of the histogram will be along the lane line. Along the curves, the lane line will be spread across the x-axis leading to a reduced peak in the histogram. A visual representation can be seen in FIGURE. A combination of the two previous factors led to the outlier seen in FIGURE.

3.8 Pipeline-Optimierungen

One of the obstacles encountered during the creation of the image processing pipeline, is that the image would be processed too slowly. The initial implementation of the pipeline ran at approximately 15 Hz, which was decided to be close to the lower bound of our chosen tolerance for the processing frequency. In order to improve the processing speed of the pipeline to an acceptable level, only 1 optimization was needed. While further optimizations are possible, they were considered to be out of the scope of the assignment and are therefore not discussed here.

The Python program cProfiler was used in order to tabulate the function calls of the pipeline, as well as the runtime of each function call. As the pipeline is a deterministic program, an simple runtime improvement is to cache the output of expensive function calls into memory. Therefore during all subsequent function calls, the cached result is returned instead of running the calculation again. As an example, the perspective transformation matrices are an expensive calculation, which when cached led to a speed improvement of the pipeline.

In order to prove the speed improvement of the pipeline, an experiment was implemented. First, the pipeline is run through once and the runtime ignored as the speed improvement is only relevant to subsequent runs of the pipeline. Then, on the second run of the pipeline, the runtime is measured for both variants, with and without caching the expensive function calls from the first run. The experiment is then run 100 times and

the result tabulated. The arithmetic mean of the runtimes for both variants are then compared to one another to measure the speed improvement. The values are in TABLE, showing that by caching the expensive function calls, the pipeline processing speed improved by VALUE

Kapitel 4

Experiment

4.1 Hardware Setup

The CoRoLa car platform is composed of a Raspberry Pi, a fisheye lens camera, a DC motor and a motor controller to control the vehicle speed and a servo motor to control the steering angle of the vehicle. The Raspberry Pi sends PWM signals to the motor controller and the servo controller in order to control the vehicle. Using Robot Operating System 2 (ros2), the Raspberry Pi is connected over the network to a central computer. The Stanley controller runs on the central computer, which processes the data from the camera and sends the output of the Stanley controller to the vehicle.

4.2 Experiment Setup

In order to compare the Stanley controller and the sliding window method with the PID controller, 4 test criteria are chosen: maximum acceptable velocity and maximum offset angles, as well as the root mean square values of the vehicle offset and controller outputs.

4.2.1 Höchstgeschwindigkeit

In order to measure the maximum acceptable velocity, vehicle drives around the track at a given forward velocity. If the behavior of the vehicle is seen as äcceptable", then the vehicle is stopped and the velocity is increased. This process repeats until the behavior of the vehicle is no longer äcceptable". Äcceptable"behavior of the vehicle is defined empirically under the following two criteria: ability to follow the lane line and an absence of observed oscillations.

4.2.2 Maximal möglicher Fehlerwinkel

The maximum offset angles are defined as the maximum angles at which the vehicle is able to find the path again, on the left and right of the lane line respectively. In order to measure this, the vehicle is placed collinear with the lane line with the motor controller disabled. The vehicle is then turned clockwise, with the output of the respective controller displayed in realtime. The vehicle is turned until the output of the controller becomes either chaotic or the output is consistently zero. The experiment is then executed for the Stanley Controller and PID Controller. The occurence of chaotic output is nondeterministic, however empirically it is when the vehicle cannot detect the lane line reliably. An output of zero also means that the vehicle cannot detect the lane line at all.

4.2.3 Abstand von der Linie

The measurement of vehicle offset is conducted through the following two experiments. First, the vehicle starts driving around the track. Then using an overhead camera, video of the vehicle driving around the curves is recorded from a top down view. The vehicle is recorded at differing forward velocities. For the first round of recordings, the vehicle is controlled by the PID controller and for the second, it is controlled by the Stanley controller. This experiment is then repeated with the camera recording video of the vehicle driving along the straights instead.

4.2.4 Reglerausgang

In order to measure the controller output of the two controllers, the vehicle is commanded to drive around the track at least 3 times. During the drive, the controller output is recorded. After the drive, the data is graphed over time and a subset of the data is selected. The subset of the data is selected under the following criteria: periodicity and absence of outliers.

Kapitel 5

Conclusion