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

Der Stanley-Regler ist ein nichtlinearer Regelalgorithmus, der 2005 von der Stanford University entwickelt wurde. Er wurde entwickelt, um ein zweiachsiges Fahrzeug zu steuern, wobei nur der Ausrichtung und der Querfehler des zu verfolgenden Weges verwendet werden. Für die Dauer dieser Arbeit wird jedoch nicht das kinematische Modell eines zweiachsigen Fahrzeugs verwendet, sondern ein Modell, das auch die Dynamik des Lenkwinkels berücksichtigt.

Der Stanley-Regler ist ein Pfadfolgerregler im Gegensatz zu einem Trajektoriefolgerregler. Der Stanley-Regler ist als Querregler konzipiert, so dass das Fahrzeug auf der Bahn bleibt, aber keinen Einfluss auf die Geschwindigkeit hat, mit der es der Bahn folgt. Die Vorteile dieses Ansatzes liegen darin, dass die Geschwindigkeit des Fahrzeugs flexibel gewählt werden kann, so dass der Ingenieur, abgesehen von dynamischen Effekten, jede für seine Anwendung erforderliche Geschwindigkeit wählen kann.

Der Stanley-Regler wird durch die folgende Gleichung dargestellt:

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

wobei u das Ausgangssignal des Reglers, theta der aktuelle Ausrichtung des Fahrzeugs, θ_d der Ausrichtung des Pfads, k ein Skalierungsfaktor, v die Geschwindigkeit des Fahrzeugs und e_{fa} der Spurabweichungswert vom Mittelpunkt der Vorderachse des Fahrzeugs zum Pfad ist.

Der Stanley-Regler besteht aus zwei Komponenten: eine Komponente händelt sich um den Abstand von dem Fahrzeug zu dem Pfad und die andere Komponente händelt sich mit der Ableitung des Abstands oder genauer geschreiben, die Ausrichtung.

Die erste Komponente, die um den Abstand geht, steuert die Lenkwinkelstrecke des

Fahrzeugs so, dass das Fahrzeug in der Richtung des Pfads je mehr lenkt, desto weiter das Fahrzeug von dem Pfad ist. Diese Komponente ist durch $\arctan\left(\frac{ke_V}{v}\right)$ dargestellt.

Die zweite Komponente steuert die Lenkwinkelstrecke so, dass sie parallel zu dem Pfad bleibt. Diese Komponente ist durch $\theta - \theta_d$ dargestellt.

Zussamen arbeiten Komponenten miteinander, sodass das Fahrzeug auf den Pfad kommt.

2.2 Simulation

Um das Verhalten des Stanley-Reglers auf dem ferngesteuerten Fahrzeug zu untersuchen, wurde eine Simulation des Regelkreises verwendet. Die Simulation besteht aus drei Hauptkomponenten: erstens die Bahn, der das Fahrzeug folgen muss, zweitens der Stanley-Regler und drittens das Fahrzeugmodell. Ein Differentialgleichungslöser wird verwendet, um die Zeitreihen des zweiachsigen Fahrzeugs zu analysieren.

Der Pfad, dem das Fahrzeug folgen muss, ist eine virtuelle Version der realen Strecke, die das Fahrzeug benutzen wird. Eine visuelle Darstellung des Weges ist ABBILDUNG. In der Simulation wird der Weg durch die folgende kontinuierliche statische Funktion dargestellt: —-, mit dem Eingang t und dem Ausgang (x, y, h). Diese Funktion wird dann für eine beliebige Anzahl von t-Werten diskretisiert, um ein Array von (x, y, theta)-Werten zu erstellen. Dieses Array wird dann in den Differentialgleichungslöser eingespeist, der den Stanley-Regler und das Fahrzeugmodell kapselt.

Bei dem verwendeten Fahrzeugmodell handelt es sich um das Modell eines Hinterachsfahrzeugs. Dieses Modell wird durch die folgende nichtlineare Zustandsraumdarstellung dargestellt:

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

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

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

$$\dot{\phi} = \phi. \tag{2.5}$$

Die Zustandsvariablen des Modells sind x_H, y_H, ϕ und θ, wobei x_H und y_H die XY-Koordinaten des Mittelpunkts der Hinterachse sind und θ der Kurs des Fahrzeugs ist. Die Eingangsvariablen des Modells sind v und ϕ_d, wobei v die Geschwindigkeit des Fahrzeugs und ϕ der Ausgang des Stanley-Reglers ist. Der Lenkwinkel ϕ wird durch eine lineare Differentialgleichung erster Ordnung mit der konfigurierbaren Zeitkonstante T dargestellt. Das Verhalten des Stanley-Reglers bei dynamischen Einflüssen auf den Lenkwinkel ist von großer Bedeutung, da die globale asymptotische Stabilität des Reglers nur für das kinematische Modell mit federndem Lenkwinkel nachgewiesen wurde.

Da der Stanley-Regler den Mittelpunkt der Vorderachse für den Querspurfehler benötigt,

muss dieser aus der Hinterachse berechnet werden. Der Mittelpunkt der Vorderachse wird über die folgende statische Funktion aus den Zustandsgrößen berechnet:

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

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

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

wobei xf und yf die x/y-Koordinaten dieses Mittelpunkts sind.

Wie bereits erwähnt, besteht der Stanley-Regler aus mehreren Teilen, die berechnet werden müssen. Der Steuerkurs ist eine Zustandsvariable und daher immer verfügbar. Für den Kreuzspurfehler und den Kurs muss der richtige Bahnpunkt gewählt werden. Der Stanley-Regler verwendet den nächstgelegenen Bahnpunkt, um den erforderlichen Kurs und den aktuellen Querfehler zu bestimmen. Für die Simulation wird dieser durch die Suche nach dem Punkt mit dem geringsten Abstand zum Vorderachsmittelpunkt bestimmt. Der Algorithmus wird wie folgt dargestellt: —. Durch diesen gefundenen Punkt wird der Kursweg bestimmt. Der Querabweichungsfehler wird dann durch die folgende Gleichung bestimmt: —. Diese Gleichung stellt das Skalarprodukt des Vektors senkrecht zum Steuerkursvektor und des Vektors von der Vorderachse zum Bahnpunkt dar. ABBILDUNG ist eine visuelle Darstellung dieser beiden Vektoren. Der Kurs- und der Spurabweichungsvektor werden dann in den Stanley-Regler eingegeben, wobei der Ausgang des Reglers in das Fahrzeugmodell eingespeist wird.

Da der Stanley-Regler nur einen Lenkwinkel ausgibt, wird die Geschwindigkeit als konstant angenommen. Ein High-Level-Diagramm des Regelkreises ist in ABBILDUNG dargestellt.

Die Simulation wird dann für eine ausgewählte Zeitspanne durchgeführt.

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

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

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

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

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

At this 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 multiple of the image width.

Second, the x-coordinate of the midpoint of the highest window is set to the peak of the histogram from the previous stage. The number of white pixels are then counted inside the window. If the number is above a chosen threshold, the mean of the x-coordinates of the contained pixels.

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.

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, therefore as there is no global frame for the vehicle, the heading of the vehicle is chosen to be 0!!!!!! at all times.

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 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

Kapitel 5

Conclusion