Proj-lfi-ml: Final Report

The Final Result

Our approach in yellow light: https://vimeo.com/380665855

Our approach in low light: https://vimeo.com/380665899

Our approach without external light: https://vimeo.com/380665936

see the operation manual to reproduce these results.

Mission and Scope

The Mission of this project is to make the current lane following algorithms more robust regarding different lighting conditions with the help of a machine learning based algorithms.

Motivation

The currently implemented lane following algorithm in Duckietown is not robust against lighting changes. Especially in difficult lighting situations the Duckiebots often fail to estimate their pose correctly. This leads to wrong controller inputs, which results in Duckiebots not driving in the middle of the lane, crashing into each other or going off the road.

The current implementation of lane following is based on the following pipeline. Using k-mean clustering the pixels received by the camera are clustered into clusters of similar colors and they are matched against an expected color map. Based on this matching an affine color transform is applied. The transformed image is used to extract the lane segments using an and gate between an HSV color thresholding filter and a canny edge detection. The extracted lane segments are then reprojected into real world coordinates. With the knowledge about the structure of Duckietown the segments can vote for a pose. This is processed by a Bayesian filter to estimate the current relative deviation and Heading to the centerline. With two separate PI controllers the error in deviation and heading is being minimized. [1]

For more information please refer to the Duckietown duckumentation

Opportunity

The lane segmentation performs poorly under difficult lighting situation. Reasons for this could be difficulties in edge detection or HSV thresholding for those lighting situations. This leads to wrongly reprojected lane segments and subsequently wrong pose estimations.

---

[1] Paull et al, 2017, Duckietown: An Open, Inexpensive and Flexible Platform for Autonomy Education and Research

We use a completely different approach. We replace the whole lane following pipeline except of the controller with a convolutional neural network. The input of the network is the unrectified compressed image stream of the Duckiebot. The output of the network is the estimated pose of the Duckiebot. To achieve this goal we take advantage of the Duckietown localization system to retrieve the ground truth pose for training.

Definition of the problem

Goal

Our final objective is to implement a more robust lane following algorithm than the currently used algorithm on the Duckiebot.

Assumptions

We assume that we have no traffic and no intersection to navigate on. Further, we assume Duckietown compliant roads.

Metric

We compare our own lane following algorithm to the current lane following algorithm in terms of robustness and reliability in different lighting conditions.

Contribution / Added functionality

To tackle the problem stated above, we propose a supervised learning approach using a deep neural network (CNN) architecture which can estimate the pose (heading and distance relative to the middle of the lane) of the Duckiebot based on the image stream of the Duckiebot. For the training of the CNN we use the ground truth pose of the Duckiebot retrieved from the Duckietown localization system. The computed pose is the input of a PID controller.

The project consists of three different parts:

- Data Acquisition and Labeling
- Model Design and Training
- Implementation

In the following is a detailed explanation of the three parts

Data Acquisition and Labeling:

We acquired the data with the localization system and labeled the data with the framework Duckietown world. In general, the localization system works well but we encountered some

issues while handling a huge amount of data. If the rosbag recordings are too big, several messages are dropped during the recording and the bag files can't be correctly post-processed because of the huge memory requirements of the post-processing. To overcome this issue, we split the recording into smaller ones of 15 seconds duration and increased the rosbag buffer size to 2048 MB. The buffersize can be increased with the -b tag (e.g. rosbag record –b 2048 -a) and the recordings can be split with the --split tag (e.g. rosbag record –b 2048 --split -a). If the rosbag file is already recorded without any split, our bash file (see our github repo) can be used.

Since the post-processing and optimization is tedious for many separate bag files we implemented two bash scripts to ease the post-processing and optimization. The bag files need to have the convention <bagname>_<sequence number>.bag (e.g recoring1_1.bag, recoring1_2.bag, recoring1_3.bag) for the post-processing and processed_<bagname>_<sequence number>.bag (e.g processed_ recoring1_1.bag, processed_ recoring1_2.bag, processed_ recoring1_3.bag) for the optimization.

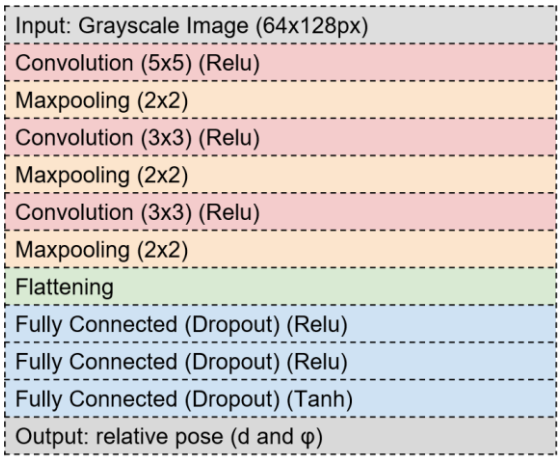
For the training we need the recorded images of the Duckiebot as jpg images and their respective timestamps. The images are stored in the recorded rosbag file and need to be extraceted. We use a simple python script to initialize a rosnode, subscribe to the image stream and convert the image stream to jpg images and stores the timestamps in a csv file.

For training neural networks using a supervised learning approach, the data must be labeled in order to be trained on. This is done with the help of the Duckietown-world package and our custom module. The module reads in the global pose for every timestamp of the trajectory that was generated by the localization system and computes the relative pose, specifically, the Duckiebots position and heading relative to the middle of the lane. It then matches the timestamps of the pose with the image frame timestamps. If the timestamps are not exactly the same, the pose is interpolated to match the timestamp of the image frames. For each image frame with it's respective pose, labels are added. The labels are the center distance to the middle of the lane (**NOT** middle of the tile), the relative heading and the tile.

Model Design:

The model architecture used is a Convolutional Neural Network (CNN). It consists of multiple convolutional layers, followed by multiple fully connected layers. This type of architecture is currently state of the art for image classification and regression problems.

The CNN architecture implemented has 3 convolutional layers. Each convolutional layer is followed by a maxpooling layer as well as by a batch normalization layer. After the 3rd convolutional layer the tensor is flattened and reduced by 3 fully connected layers up to 2 output nodes. Except the last layer, all the layers use ReLu as the activation function. The very last fully connected layer uses a tanh activation function to reduce outliers. The feature size and the size of learnable parameter depends on the image resolution, number of image channels and cropping area.



For training the data is augmented. This results in a larger dataset, a higher robustness and reduced bias of the data. For data augmentation we used the following methods:

- Change of Contrast
- Change of Brightness
- Change of White Balance
- Image cropping
- Image patch removal

Model Training:
The model was then trained using the labeled dataset collected before. The training was done on an IDSC Server, which provided multiple GPUs and CPUs for training. In average the model training converged after 200 epochs and was trained on 1 GPU and 16 CPU cores (to speed up data loading and preprocessing). This took about 6-8hrs per training.

The model was then evaluated with unseen datasets and later in directly in the real implementation.

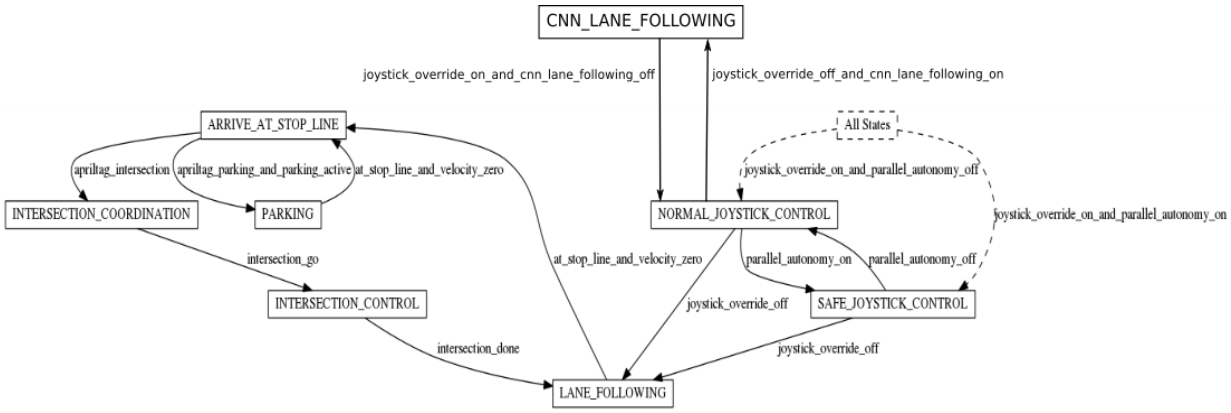
Implementation
To run the machine learning based lane following three different docker images

- cnn_node
- dt-core
- dt-car-interface

need to be built and run on the Duckiebot (dt-core and dt-car-interface) and on the computer (cnn_node)

The machine learning based lane following works as follows: A rosnode named cnn_node is initialized. The node subscribes to /camera_node/image/compressed. When the node receives a new image message the image is turned into grayscale and cropped. The cropped grayscale image is the input to the convolutional neural network (CNN). The ouput of the CNN is the relative pose, i.e., distance to middle lane and relative heading of the Duckiebot. The relative pose is used to computed the control signal (velocity and angular velocity) of the Duckiebot using a PID controller. Eventually, the control signal is published in the topic cnn_node/car_cmd. We implemented the CNN using PyTorch

In dt-core and dt-car-interface we added a state called CNN_LANE_FOLLOWING. The state transition from NORMAL_JOYSTICK_CONTROL to CNN_LANE_FOLLOWING is toggeled if joystick_override_off_and_cnn_lane_following_on is true. The state transition from CNN_LANE_FOLLOWING to NORMAL_JOYSTICK_CONTROL is toggled if joystick_override_on_and_cnn_lane_following_of is true



Apart from the the CNN_LANE_FOLLOWING state dt-core and dt-car-interface are the same as the official Duckietown dt-core and dt-car-interface.

The controller used for this project makes use of the already implemented PI-controller in dt-core. It is a tuned, cleaned-up and asymmetric PID-controller. It weighs deviations to the outside of the tile more than towards the middle lane.

Unfortunately, we could not implement the CNN on the Duckiebot. The problem is that the Duckiebot still runs on Python 2.7 and there doesn't existed a compiled PyTorch library for arm32v7 and Python 2.7. We tried to build PyTorch directly on the Duckiebot but after 4 days the Duckiebot still hasn't finished compiling. An alternative is onnxruntime. Unfortunately, onnxruntime requires Python 3.5. We also tried to hard code the CNN with NumPy but only the first CNN layer already requires 1.5 seconds on an i7 Laptop to be computed. The whole network using PyTorch requires 0.01 seconds on the computer. Therefore, we couldn't test the network on the Duckiebot.

Formal performance evaluation / Results

In the beginning, we summarized the goal of our project as followed: “we need to reliably detect lanes in different lighting conditions. From the detected lanes we need to accurately estimate the Duckiebots relative position and orientation. With the estimate, control the Duckiebot more robustly than with the current implementation.”

The performance should be evaluated by the following predefined criteria:

- AIDO Quality Score
- Error in heading and position
- Performance in different environments

In the following paragraph, the results are evaluated, discussed and compared with the current implementation.

The first goal was to reliably detect lanes in different lighting conditions. The current implementation does reasonably well in bright lightning conditions. As soon as the lighting conditions are darker or as the luminance is warmer, problems start to surface. This can be explained by looking at the way the lane detection is implemented. It performs an HSV evaluation to detect the yellow middle lane of the tile and the white lanes. With difficult lighting conditions, the colors may appear different and the colors cannot be recognized anymore. Before we could tackle the problem of lane detection, the biggest challenge was setting up a framework where we can acquire data, label it and train a neural network on. Setting up these processes were the most time consuming, as similar projects have not been done on Duckietown yet. After creating the framework, we approached the problem of lane detection by not training our neural network on a color image but on shapes and features. The image is read in as a grayscale image. We also gathered data with different lighting conditions, such as warm and dark light. This leads to the result that our neural network is able to detect lanes regardless of lighting condition and performs an accurate pose estimation.

The second goal was to accurately estimate the Duckiebots relative position and orientation. In the current implementation, the detection of the lane and the estimation of the pose is split up in multiple nodes. Our goal with the neural network was to replace the whole pipeline with one node. This implies that the result of the goal is tied together with the target to correctly detect lanes and it cannot be measured which task is performed better. Only the correct pose estimation is a way to determine the correct outcome. Testing showed that the relative heading detection works satisfyingly well while the detection of the distance to the middle of the lane is not as good. The problem here lies in curves. Because the camera looks a certain distance ahead, the estimation of the distance to the middle of the lane proves difficult.

The third goal was to be able to successfully convert the pose estimation into wheel commands, resulting in a good lane following. The PID-controller we implemented works well in real

Duckietown situations. It can however be improved by integrating a kinematic model of the Duckiebot into the controller.

Our project also took part in AIDO (AI driving olympics), which was an important factor in our measurement of our success. In order to compete in the event on a real track, a simulator test has to be passed. Without ever training on simulator data, our implementation managed to successfully pass the simulator test. This speaks to the quality of our neural network, performing in an environment it has never seen. In the real evaluation at AIDO, we managed to be placed 7^{th} out of 34 competitors.

Future avenues of development

We are extremely confident that our CNN can be improved to work even better than it does now. The camera calibration can be integrated to undistort the images, which could result in better pose detection learning. What can also be tried is to insert a Birds Eye View transformation. This approach does not guarantee better results but can be tried.

The previous poses of the Duckiebot could help the CNN to compute the current pose of the Duckiebot. This can be achieved with a 3d convolution which would also input the previous images recorded by the Duckiebot into CNN. Also the previously estimated state could be integrated into the current implementation of the CNN.

Some recurrent neural network (RNN) layers could also help the network to predict the pose. In a RNN each node has an internal state which depends on the previous input. This would ultimately lead to a better estimation of the current state.

A major issue of the system is the lag to stream data from the bot to the computer and back. The Duckiebot would be more robust if the Duckiebot has the CNN directly implemented. Unfortunately, first the Duckiebot needs to run on Python 3.5 or higher to use a prebuilt PyTorch library or onnxruntime. Furthermore, the computing power would need to be significantly higher than with the current setup.