Simulation-Based Autonomous Driving in Crowded City (SS23 - Team08)

Aristotelis Tsoutsanis and Pascal Henschke

***Abstract*—This report presents the methodology and key components of our self-driving project, aimed at achieving safe and efficient autonomous navigation in urban environments. We detail the foundational aspects of our project, including the Steering Model, Data Collection, Preprocessing, and Augmentation techniques. Additionally, we delve into the YOLOv5-based Object Detection, providing insights into its architecture, model structure, and advanced data augmentation strategies. The Traffic Light Classifier, a pivotal component of our project, is thoroughly explained, outlining the precise steps involved in detecting and classifying traffic light colors, crucial for our vehicle's real-time response to traffic signals. Furthermore, we introduce integration rules encompassing vehicle detection, traffic light classification, and brake system management, pivotal in optimizing the efficiency, safety, and regulatory compliance of our autonomous vehicle in complex urban traffic environments. Our experimental results demonstrate the capabilities of our system, including successful intersection navigation, accurate traffic light recognition, and prompt response to red lights. However, we acknowledge certain challenges such as real-time prediction lag, delayed braking at red lights, and high-speed scenarios, which are subjects of ongoing research and development. In summary, our self-driving project employs a multifaceted approach to address the complexities of urban driving scenarios, combining cutting-edge technologies and meticulous rule-based systems to achieve an autonomous driving system.**

***Impact Statement*—Our self-driving project has the potential to improve urban transportation with safe and efficient autonomous navigation. Our advanced Steering Model, YOLOv5-based Object Detection, and Traffic Light Classifier contribute to an adaptable autonomous driving system. Beyond autonomous vehicles, this work could inform the development of intelligent transportation systems to reduce traffic congestion and improve urban traffic flow. As we refine our system, we aim to make autonomous vehicles coexist seamlessly with human-driven ones, transforming urban transportation for the better.**

Aristotelis Tsoutsanis and Pascal Henschke are both master students of Computer Science at the Technical University of Munich (e-mails: a.tsoutsanis@tum.de and pascal.henschke@tum.de).

I. Introduction and Motivation

Autonomous driving has emerged as a transformative technology with the potential to revolutionize transportation systems. One of the leading pioneers in this field is Waymo, which has been at the forefront of developing self-driving vehicles for urban environments. Urban areas present unique navigational challenges due to their complex and dynamic nature, where crowded city streets demand a high level of precision, adaptability, and safety in autonomous vehicles.

In the pursuit of overcoming these challenges, this project aims to develop an innovative solution using a simulator-based approach that leverages vision-based techniques. By integrating state-of-the-art deep learning models and computer vision algorithms, this project aspires to enhance the capabilities of autonomous vehicles in urban settings, ultimately leading to safer and more efficient transportation systems.

The motivation behind this project is to address the complex navigational hurdles that autonomous vehicles encounter in urban environments. While autonomous driving technology has made significant strides, the intricacies of crowded city streets present ongoing challenges. These challenges include intricate traffic patterns, unstructured and unpredictable pedestrian behavior, diverse road users, and the need for real-time decision-making in response to rapidly changing scenarios.

The primary goal of this project is to develop a comprehensive solution that enables autonomous vehicles to navigate crowded city streets with an emphasis on safety and efficiency. By utilizing a vision-based approach, the project seeks to equip vehicles with the ability to perceive their surroundings accurately, identify and classify various objects such as vehicles and pedestrians, and make informed decisions, including responding to traffic lights and other critical signals.

II. Related Research

This section presents some key research that is relevant for the autonomous driving task at hand. Broadly seen, there are two main approaches to autonomous driving. First, the end-to-end approach in which a machine learning model directly predicts the car's controls (such as throttle, steering angle, or a trajectory) given sensor inputs (e.g., cameras or lidar). Second, the modular approach in which various subsystems fulfill different purposes such as object detection, depth estimation, trajectory planning, and more. This approach often involves some amount of hard-coded rules to make control decisions based on the information synthesized from all the subsystems. This approach carries the problem that it is extremely difficult to cover (and maintain) all the possible edge cases that an autonomous agent might encounter. Because of such issues with the modular approach, the end-to-end approach is increasingly gaining popularity in autonomous driving applications. As we utilize mostly end-to-end methods in our own methodology, we focus here on presenting research that focuses on end-to-end methods.

One of the first papers showing appreciable success in real-world end-to-end autonomous driving is the Nvidia paper [1]. In this paper, the authors use a simple convolutional end-to-end model that, using image input from a single front camera, predicts the vehicle's steering angle. Their model only has 250k parameters and performs quite well in simple driving situations. A detailed view of the model architecture can be seen in Figure 2. For the training of their network, they use

input from two more offset front facing cameras to provide training data that shows recovery from driving mistakes and sub-optimal positioning on the road. This helps with the general issue that the training data that is recorded from an expert driver does not contain much information on recovering a proper vehicle trajectory.

Another basic problem in autonomous driving is the ambiguity of actions. When a car approaches an intersection for example, it is not clear from visual input whether it should go left, straight, or right. The same image can therefore be associated with completely different steering angles in the training data, impeding the model's ability to navigate such ambiguous scenarios. To solve this problem, and to allow navigational commands to be provided as model input, Codevilla et al. [2] introduced **CIL**, a model that takes a high-level navigational command as input in addition to sensor input. These commands are very basic instructions such as "go left/right/straight at next intersection" or "follow lane". To accommodate such a command input, they propose two models:

(i) a concatenated architecture that simply concatenates a latent representation of the navigational command input with a latent representation of the sensor outputs (e.g., the flattened output of the convolutional layers processing the image).

(ii) a branched architecture that has multiple different fully-connected model heads for each navigational command, where the command acts as a switch between these different branches. This allows each fully-connected layer to specialize for a specific driving task associated with the navigational command.

They find that the branched architecture shows better driving performance in the CARLA driving simulator and on a toy physical system. Another novel point in their paper is the fact that they not only predict lateral (i.e., steering) but also longitudinal commands (i.e., throttle and brake), which is of course a requirement for true self-driving. In all their architectures, they use a simple convolutional architecture quite similar to the Nvidia model for the image perception module.

In a follow-up paper, Codevilla et al. improve their branched-conditional architecture to create a model they call **CILRS** [3]. Here, they use a ResNet34 for the perception module. In addition, they use this perception module to predict the speed of the vehicle. They find that this has a driving-performance enhancing effect because it has a regularizing effect and allows the perception module to learn speed-related features. Furthermore, they use an L1 loss instead of their previously used MSE loss, as other research has found that the L1 metric is more correlated with actual driving performance [4].

As of September 2023, the state-of-the art in end-to-end self driving, according to the ranking on the CARLA 1.0 Leaderboard (1st place), is **ReasonNet** by Shao et al [5]. ReasonNet uses a transformer-based architecture comprised of three major components: A perception module, a temporal reasoning module and a global reasoning module. The perception module processes the sensor input and constructs birds-eye-view features. The temporal reasoning module is responsible for storing and processing historical information about the driving scene. Its primary intended purpose is to generate accurate predictions regarding the behavior of other actors in the environment. The global reasoning module is focused on modeling the interactions and relationships among objects and the surrounding environment. Its main objective is to identify adverse events, especially occlusions (situations where relevant objects are hidden or obstructed from view), and enhance the overall perception performance of the autonomous driving system. Importantly, ReasonNet uses an extensive set of sensors as input, including cameras all around the car and lidar. Furthermore, privileged information such as birds-eye-view maps are used to train the model, making their approach infeasible for our own approach in the simple vision-based simulator.

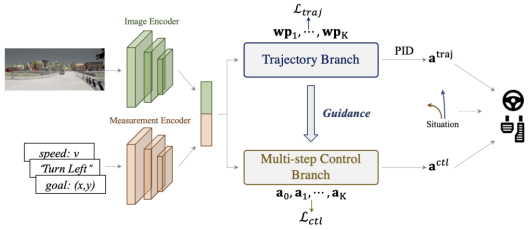

Fig. 1. Overview of Trajectory-guided Control Prediction (TCP). The encoded features are shared by the trajectory and multi-step control branch. The trajectory branch provides per-step guidance for multi-step control prediction. Outputs from two branches are combined according to our situation based fusion scheme to generate the ultimate control actions.

More relevant to our task at hand is the **TCP** framework by Wu et al. [6]. This model is currently third place in the CARLA benchmark, but contrary to ReasonNet it uses only a single front camera image as input and no privileged information during model training. When constructing an end-to-end autonomous driving system, there are broadly speaking two choices one can make regarding the final lateral model output. First, the model can directly output a steering angle and second, the model can output a predicted trajectory for the ego-vehicle (intuitively waypoints on the road that show where the car should go). In the second case, an additional controller that produces steering commands to fulfill this planned trajectory is needed. While CIL and CILRS use the first method, ReasonNet uses the second. What is novel about TCP is that it uses a mix of both. Starting from the observation that direct-control prediction approaches work best when performing sharper turns, while trajectory-planning approaches work best with less sharp turns, the authors build a kind of multi-task learning framework in which the final control command depends on both a a multi-step-control branch and a trajectory-planning branch. An overview of the architecture is shown in Figure 2. The TCP architecture is composed of three primary components: an initial input encoding stage and the two branches. An image and a measurement encoder produce latent representations of the front camera image, and of other vehicle measurements as well as high-level navigational commands. These latent representations are concatenated (as in the concatenated architecture of the CIL paper) and shared between the two branches. Notably, the control branch employs a multi-

step prediction design, guided by insights from the trajectory branch through the use of an attention map. Both branches of the TCP framework leverage Gated Recurrent Units (GRUs) to capture and process temporal information. This enables the TCP framework to make informed decisions based on the evolving context of the road environment. To produce the final control commands, the TCP architecture adopts a situation-based fusion scheme in which the outputs of both branches are combined into a weighted average, with the weights depending on whether the vehicle is currently turning (more weight on control branch) or not (trajectory specialized).

III. Our Methodology

A. Steering Model - Udacity Simulator

In our pursuit of developing a robust steering model for our autonomous driving system, we explored two distinct architectures within the Udacity Simulator environment: the Nvidia Model [1] and a customized ResNet50 variant [7]. These models were trained to predict steering commands that minimized the mean squared error compared to human driver steering commands or adjusted commands for off-center and rotated images.

1) Nvidia Model: Our implementation of the Nvidia Model, inspired by the original paper, adheres to a deep neural network architecture designed specifically for steering angle prediction.

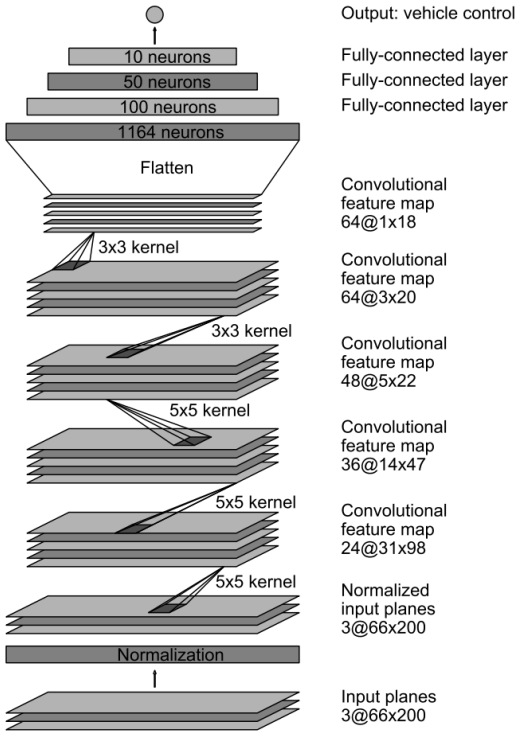

Fig. 2. CNN architecture. The network has about 27 million connections and 250 thousand parameters.

The key characteristics of this architecture include (Figure 1.):

- **Normalization Layer:** The network begins with a normalization layer that processes the input image. This normalization scheme is hard-coded and remains fixed during the learning process. The inclusion of this layer enables flexibility in adapting the normalization scheme to the network architecture while benefiting from GPU acceleration.
- **Convolutional Layers:** The architecture comprises five convolutional layers, chosen empirically after conducting experiments to optimize layer configurations. The first three convolutional layers employ strided convolutions with a 2×2 stride and a 5×5 kernel, while the last two use non-strided convolutions with a 3×3 kernel size. These layers are primarily responsible for feature extraction.
- **Fully Connected Layers:** Following the convolutional layers, three fully connected layers are utilized to produce the final steering control value. These fully connected layers function as a controller for steering, although in an end-to-end trained system, the distinction between feature extraction and control functions is less defined.

Our rigorous experimentation revealed that the Nvidia Model consistently outperformed other models, delivering superior results in steering angle prediction for the Udacity environment.

2) ResNet50 Layers: In contrast to the Nvidia Model, we explored a ResNet50-based architecture for steering angle prediction. This model capitalizes on the capabilities of deep residual networks and incorporates the following:

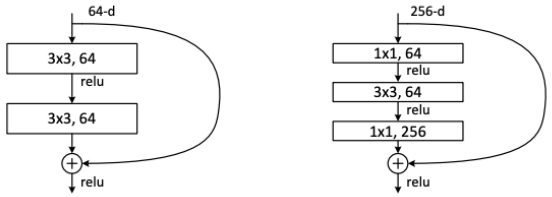

Fig. 3. A deeper residual function F for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet34. Right: a "bottleneck" building block for ResNet-50/101/152

- **Bottleneck Blocks:** We replace the 2-layer blocks found in the original 34-layer ResNet architecture with 3-layer bottleneck blocks, resulting in a 50-layer ResNet variant (Figure 2). These blocks follow option B for increasing dimensions and contribute to a model with 3.8 billion FLOPs (floating-point operations per second).
- **Pre-trained Model:** To harness the representational power of neural networks, we initially employ a pre-trained ResNet50 model trained on the ImageNet dataset. Subsequently, we fine-tune this pre-trained model using our captured driving images to adapt it to the specific task of steering angle prediction.

While both the Nvidia Model and the ResNet50 variant were explored, the Nvidia Model's superior performance led us to favor its adoption as the primary steering model for our autonomous driving system. The Resnet50 being outperformed despite its much larger size is likely due to the simplicity of the

Udacity environment, where a model as large as the ResNet is more likely to overfit.

3) Video Demonstration: To showcase the effectiveness of our Nvidia Model and its applicability, we have prepared a video demonstration of our self-driving vehicle navigating various scenarios within the Udacity Simulator. You can view the video by following this link: https://drive.google.com/file/d/1j1uXR9_idY1suzlLYG9j9bgB7_czIzk4/view?usp=sharing.

This demonstration highlights the capabilities of our steering model, offering a glimpse into the precision and reliability it brings to our autonomous driving system.

B. Steering Model - TUM Simulator

When designing the architecture for the TUM simulator, the simulator setup and the general constraints have to be taken into account. As we only have front-camera images, no access to privileged information within the simulator, no access to an automated expert driver (as in Carla for example), and in general limited resources in terms of compute and human effort, we decided to stick to an architecture that is simple and lightweight to allow faster training and iteration. To reduce complexity, and to spare the effort of building an appropriate controller for a closed-source simulator, we adopt a direct-control approach and not a trajectory-planning approach. As the urban environment in the TUM simulator largely consists of intersections, the model's capability to handle the traffic scenarios present at intersections is essential. Because of the previously mentioned training data ambiguity problem at intersections, it is essential to incorporate high-level navigational commands into the model. We do this by adopting the concatenated command-conditional architecture from the CIL paper. Our method for obtaining data labelled with navigational commands is detailled in the Data Collection section. Because smooth manual longitudinal control of the vehicle in the simulator (which only ran with very low FPS on the author's computers impeding data collection) was extremely difficult, we opted for our model to only take care of the vehicle's steering, while adopting a rules-based approach to control the vehicle's speed.

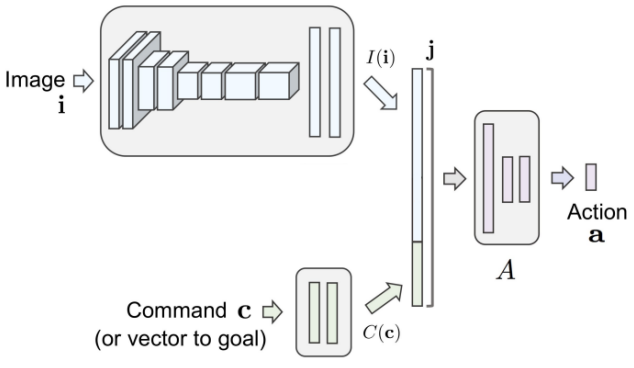

Fig. 4. High-level overview of our model for the TUM simulator, which mimics the command-concatenated architecture from the CIL paper.

Model Architecture: Our high-level model architecture is sketched out in Figure 4. For the image encoder module that produces a latent representation of the front-camera image we use an improved version of the Nvidia model. Following the Nvidia model, our image encoder includes five convolutional layers with the number of feature maps increasing from 24 in the first convolutional layer to 64 in the last convolutional layer. In contrast to the Nvidia model, we apply BatchNorms after each convolution, and use ReLU activation functions as opposed to the original ELU activations. After flattening the output of the last convolution, we apply a dropout with a probability of 0.5 to aid in regularization and prevent overfitting. The high-level navigational command is one-hot-encoded and upsampled to a vector of length 128. Then a dropout of 0.25 is applied and a ReLU activation is applied. The resulting command encoding is concatenated with the image encoding and run through three fully-connected layers with ReLU activations.

C. Data

In this section, we describe the data pipeline used in our autonomous driving system, including data collection, preprocessing, and augmentation techniques.

1) Data Collection: Our data collection process involves manually driving the car using a mouse to provide steering angle values as labels. Due to performance issues in the TUM simulator, accurate manual control of the vehicle was hard, making the collection of flawless high quality data more difficult. Our data collection process included the following steps:

- **Manual Steering:** During data collection, we manually steers the car by controlling the mouse. Mouse input is needed to get a fine-grained steering angle. This fine-grained steering input is essential for the training of our steering prediction model.
- **Command Labels:** As discussed before, navigational command labels are needed. As manual human labelling of each data point is infeasible, and as the simulator does not support the labelling of high-level navigational commands through the use of indicators, we realize this labelling by recording multiple separate runs, in which we try to exclusively follow one specific high-level navigational command. The set of navigational commands we introduce is: (1) turn left, (2) turn right, and (3) go straight.
- **Separate Runs:** We conduct separate runs for different navigation commands to allow easy labelling of the data with navigational commands and to ensure diverse and representative data for each driving scenario. By recording different runs in which we try to only turn left, turn right, or go straight at intersections, we can label the whole run using the appropriate command label, saving us the infeasible amount of effort of manual labelling. This separation also enhances the model's ability to generalize to different situations.
- **Recovery Runs:** Specialized "recovery runs" are executed to teach the model how to recover from challenging situations. These runs provide valuable data on regaining control when deviating from the desired path.
- **Data Balancing:** Efforts are made to balance the data by reducing near-zero angle observations. This balancing

improves the model's performance by addressing bias toward a specific steering angle.

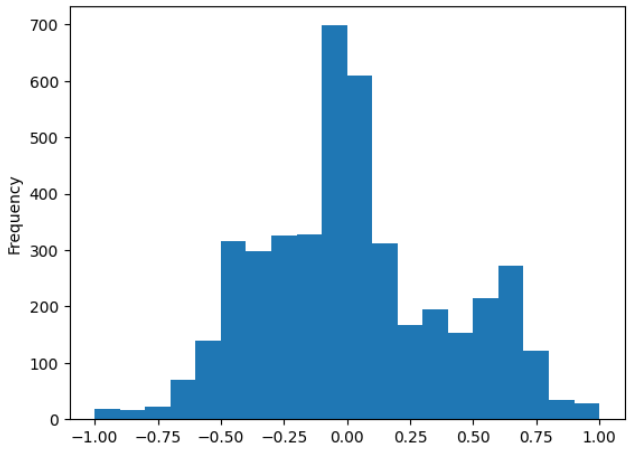

Fig. 5. Steering angle distribution after balancing the data

2) Data Preprocessing and Augmentation: Data preprocessing and augmentation techniques are applied to enhance the quality and diversity of our training data:

- **Simple Augmentations:** We apply simple augmentations to the data, including cropping the sky (as the sky is irrelevant for driving decisions), adding blur, adjusting brightness, and introducing color jitter. These augmentations simulate variations in lighting and road conditions, enabling the model to generalize better.
- **Data Balancing:** Our ongoing efforts involve achieving more accurate data balancing across the entire steering angle spectrum. This ensures that the model receives sufficient training data for all steering angles, leading to less bias and improved performance.

3) Next Steps: To further enhance our data pipeline and improve training data quality, we plan to:

- **Accurate Data Balancing:** Continue refining our data balancing techniques to ensure that the training data distribution accurately reflects real-world scenarios, addressing potential biases and improving model generalization.
- **Side Cameras with Corrective Factor:** To teach the model recovery strategies, we should utilize side cameras with a corrective factor. This means that data from side cameras is used alongside center camera data with adjustments to simulate recovery actions when the car deviates from the intended path.

In conclusion, our data pipeline plays a crucial role in training our autonomous driving system. By collecting diverse and well-labeled data, applying augmentations, and balancing the dataset, we aim to develop a robust and adaptive model capable of safe and reliable autonomous navigation.

D. Object Detection - YOLOv5 Deep Dive

1) Introduction to YOLOv5: YOLOv5 (version 6.0/6.1) stands as a significant advancement in the field of object detection, crafted by Ultralytics [8]. This algorithm empowers the accurate detection of objects within images and has profound applications in domains such as surveillance, autonomous vehicles, and image recognition. This section delves into the architecture, training methodologies, and data augmentation strategies employed by YOLOv5, offering a comprehensive insight into its practical application.

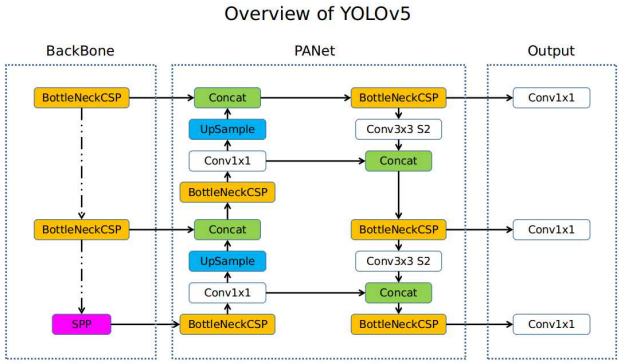

Fig. 6. The architecture of YOLOv5

2) Model Structure:

a) Backbone: At the core of YOLOv5's architecture lies the backbone, which forms the main body of the network. Unlike its predecessors, YOLOv5 adopts the New CSP-Darknet53 structure as its backbone, building upon the Darknet architecture. This modification enhances the network's capacity to capture complex features and patterns within images.

b) Neck: Connecting the backbone and the head, the neck of YOLOv5 introduces the SPPF (Spatial Pyramid Pooling Fusion) and New CSP-PAN structures. These additions enhance the network's spatial awareness and enable it to process information more effectively.

c) Head: The head of YOLOv5 is responsible for producing the final detection output. To achieve this, YOLOv5 utilizes the YOLOv3 Head, a proven component that generates accurate predictions.

YOLOv5 introduces notable improvements over its predecessors:

- Enhanced Focus Structure: The Focus structure is replaced with a more efficient 6x6 Conv2d structure, marked as an enhancement.
- Upgraded SPP Structure: The SPP (Spatial Pyramid Pooling) structure is upgraded to SPPF, more than doubling the processing speed while maintaining accuracy.

3) Data Augmentation Techniques: YOLOv5 leverages a suite of data augmentation techniques to bolster the model's generalization capabilities and mitigate overfitting. These techniques include:

a) Mosaic Augmentation: This technique merges four training images into one, fostering the model's ability to handle diverse object scales and translations effectively.

b) Copy-Paste Augmentation: A novel method that duplicates random patches from one image and pastes them onto another, creating fresh training samples that simulate varying scenarios.

c) Random Affine Transformations: This involves introducing random rotations, scaling, translations, and shearing to the training images, diversifying the dataset.

d) MixUp Augmentation: By generating composite images through linear combinations of images and their corresponding labels, MixUp augments the dataset's variety.

e) Albumentations: A powerful image augmentation library is employed to support a wide array of augmentation techniques, enhancing dataset diversity.

f) HSV Augmentation: Random variations are applied to the Hue, Saturation, and Value of the images, contributing to further diversity.

g) Random Horizontal Flip: An augmentation technique that horizontally flips images at random, increasing the variability of training data.

E. Traffic Light Classifier

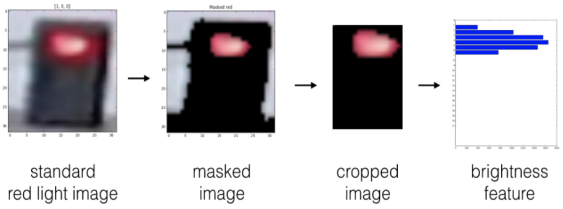

Fig. 7. A sample pipeline for creating a brightness feature (from left to right: standardized image, HSV color-masked image, cropped image, brightness feature)

Our traffic light color classifier plays a crucial role in our self-driving autonomous vehicle project designed for city environments. It ensures that our vehicles can accurately interpret and respond to traffic light signals, contributing to the safety and efficiency of autonomous driving in urban areas. Here, we provide a detailed breakdown of the steps involved in our classifier pipeline:

Object Detection: The pipeline begins with the detection of traffic lights using our advanced object detection model. This model is trained to identify traffic lights within the vehicle's field of view, providing precise bounding boxes around them. These bounding boxes serve as the regions of interest for our subsequent color classification process.

1) Image Resize (32x8): To optimize our processing speed and memory usage, we resize the cropped traffic light images to a standardized dimension of 32x8 pixels. This resizing ensures that all traffic light images have consistent dimensions, making it easier to extract and compare color information accurately.

2) Color Mask Creation: Our color classification process hinges on the creation of three distinct color masks: one for each of the traffic light colors – red, yellow, and green. These masks are designed to highlight the presence of each color within the resized traffic light image.

3) Mask Slicing: Precision is key in our color analysis. For instance, when dealing with the red mask image, we selectively slice the image to focus on the region of interest. Specifically, we extract the first 10 rows of pixels (0:10) within the image. This approach is particularly effective for red lights, as their approximate positions are known in advance, allowing us to hone in on the most relevant areas for color assessment.

4) Color Comparison: The final step of our pipeline involves a meticulous color comparison process. We compare the pixel values within the sliced color masks to determine which color is most prominent. The color with the highest intensity or prevalence within the specified region is identified as the current state of the traffic light.

By following these carefully orchestrated steps, our traffic light color classifier efficiently and accurately determines the state of traffic lights, enabling our autonomous vehicles to respond in real-time. This capability is instrumental in ensuring that our vehicles adhere to traffic regulations, make safe driving decisions, and contribute to the seamless integration of autonomous vehicles into city traffic.

F. Rules

In the pursuit of a fully autonomous driving system designed for urban environments, our project seamlessly integrates the detection of vehicles and the classification of traffic light colors with a brake system. This integration is pivotal for enhancing the efficiency, safety, and compliance of our autonomous vehicle with driving regulations.

1) Detection and Classification of Traffic Lights:

- **Vehicle Detection:** Our system excels in the detection of cars and buses within the vicinity of our autonomous vehicle. This capability is vital for understanding the dynamic traffic environment and ensuring safe interaction with other road users.
- **Traffic Light Color Classification:** In parallel, we deploy our traffic light color classifier to precisely determine the state of traffic lights at intersections. This feature is indispensable for intelligent decision-making, allowing our vehicle to respond appropriately to traffic signals, whether they are red, yellow, or green.

2) Planning Rules for Efficient Driving:

- **Brake Area Utilization:** To optimize the efficiency of our self-driving car, we employ a pre-defined brake area concept. This brake area is strategically placed in front of our vehicle and serves as a safety buffer zone. The Intersection over Union (IoU) metric is utilized to measure the proximity of moving objects to our vehicle within this brake area.
- **Red Traffic Light Response:** When our vehicle detects a red traffic light within the predefined brake area, it triggers an immediate response. The car is forced to brake to a complete stop, ensuring full compliance with traffic regulations and enhancing safety at intersections.
- **Addressing Pedestrian Traffic Lights:** While our system efficiently handles standard traffic lights, we acknowledge that pedestrian traffic lights pose unique challenges. To address this issue, we are actively working on a solution that involves classifying different traffic lights based on the size of their bounding boxes. This will enable our system to distinguish between regular traffic

lights and those intended specifically for pedestrians, ensuring accurate response strategies.
- **Throttle Control:** In addition to braking, we employ throttle control as a measure to maintain a safe and normal driving speed. If the speed of our vehicle exceeds 12 km/h, we automatically set the throttle to 0, ensuring that our autonomous vehicle operates in a low speed; avoiding the car to accelerate in high speeds and keep the FPS as high as possible.

This comprehensive integration of vehicle detection, traffic light classification, and brake system management empowers our self-driving system to navigate city environments confidently and safely.

G. Experimental Results

1) Steering Model Validation Loss: The validation loss reported for our steering model training is as follows:

- **Steering Model L1 Validation Loss:** 0.028

This validation loss metric provides some insight into the performance and accuracy of our steering model during training. Nevertheless, we have not yet been able to achieve satisfactory turning capabilities, which could be due to performance issues with the simulator.

2) Video: Autonomous Intersection Navigation: In this example, our car is able to pass an intersection. For this test, we have set a really low IoU value for our braking rule method and therefore, the car stops once it detects a car in the braking area.

The video demonstrating our autonomous car safely navigating an intersection can be viewed by following this link: https://drive.google.com/file/d/1oTfNs8idTKqM6QeNlFu-QOascBC6zgWI/view?usp=drive_link

3) Pros: Our autonomous driving system exhibits several noteworthy advantages:

- **Adherence to Intersection Rules:** The car successfully navigates intersections by following predefined rules, ensuring safety and compliance with traffic regulations.
- **Traffic Light Recognition:** Our system demonstrates the ability to accurately recognize and classify the color of traffic lights, a critical component for making informed driving decisions.
- **Red Light Compliance:** When a red traffic light is detected, our car promptly initiates braking, demonstrating a commitment to safety and adherence to traffic signals.
- **Object Detection and Braking:** Our system efficiently detects passing cars and initiates braking when necessary, thereby minimizing the risk of collisions within the brake area.

4) Cons - Issues: While our autonomous driving system showcases impressive capabilities, certain challenges and limitations of the system and the current state of the TUM simulator are worth noting:

- **Real-time Predictions Lag:** The system faces difficulties in making real-time predictions due to the inherent lag within the simulator environment, impacting response times.
- **Delayed Response at Red Lights:** Although our car stops at red traffic lights, there is a delay of 3-4 seconds before braking, resulting in the car waiting in the middle of the intersection.
- **High-Speed Scenarios:** In situations where the car is moving at high speeds and encounters a vehicle in its path, the delay in response may pose a risk of collision.
- **Pedestrian Traffic Lights:** Our system may experience confusion at pedestrian traffic lights, potentially causing the car to halt indefinitely since it consistently detects a red light.

Our ongoing efforts are directed at addressing these limitations and improving the overall performance of our autonomous driving system. Through continued research and development, we aim to enhance real-time decision-making, and ensure the system's adaptability to diverse traffic scenarios.

IV. Discussion

In this section, we evaluate our work, compare it to the current state-of-the-art, and discuss the strengths and limitations of our autonomous driving system.

A. Comparison with State-of-the-Art (SoTA)

While our autonomous driving system has demonstrated significant progress and capabilities, it's essential to acknowledge the distinctions when compared to the current state-of-the-art:

1) **Temporal Information Handling:** Our system primarily focuses on spatial information processing, neglecting the incorporation of temporal information. State-of-the-art systems often leverage advanced methods such as Transformers, 3D-Networks, or Recurrent Neural Networks (RNNs) to capture and utilize temporal dependencies. The absence of temporal information handling may limit our system's ability to predict and adapt to dynamic traffic scenarios effectively.
2) **Path and Trajectory Planning:** Unlike advanced systems that integrate sophisticated path and trajectory planning algorithms, our current system lacks this critical capability. State-of-the-art systems can navigate complex road layouts and intersections with precision, optimizing routes and ensuring efficient and safe driving.
3) **Bird's-eye View Perspective:** Some advanced systems provide a bird's-eye view of the vehicle's surroundings, offering a comprehensive understanding of the environment. Our system does not offer this feature, which can be valuable in enhancing situational awareness, especially in crowded urban settings.
4) **Model-based Longitudinal Output:** Our system primarily relies on steering angle prediction and detection-based stopping decisions. In contrast, advanced systems often incorporate model-based longitudinal control, allowing for more predictive and smoother braking and acceleration. This leads to a more comfortable and safe driving experience.

B. Strengths and Limitations

Our autonomous driving system exhibits several strengths and limitations:

1) Strengths:

- **Traffic Signal Recognition:** Our system excels in recognizing and classifying traffic signals, contributing to safe and compliant driving behavior.
- **Object Detection:** The system efficiently detects vehicles and triggers braking when necessary, enhancing safety and preventing vehicular collisions.
- **Adherence to Rules:** It reliably follows traffic rules, including stopping at red lights and responding to changing traffic conditions.
- **Navigational Capability:** Through the model's navigational command input, the basic groundwork to allow navigation is laid.

2) Limitations:

- **Temporal Information:** The absence of temporal information processing may hinder real-time adaptability in dynamic traffic scenarios. Our system's limitation regarding temporal information is that it relies solely on the most recent frame for predictions, without retaining or considering the latest information over time. Current SotA methods all incorporate some type of ML method that captures temporal information. It is also intuitive that proper driving is aided by information from the last few seconds of the driving experience.
- **Path Planning:** The system lacks advanced path and trajectory planning capabilities, limiting its navigation in complex environments. As shown by the TCP framework [6], inclusion of trajectory planning into the model can benefit driving performance, and also increase interpretability of the system.
- **Bird's-eye View:** It's worth mentioning that several prominent companies in the autonomous driving field, including Waymo and Tesla, are actively leveraging the bird's-eye view perspective in their systems. This approach significantly enhances situational awareness, especially in densely populated urban areas. So, while our current limitation related to the absence of a bird's-eye view, it's important to recognize that industry leaders have integrated this valuable perspective to overcome similar challenges effectively. The absence of a bird's-eye view perspective may restrict situational awareness, especially in densely populated urban areas.
- **Longitudinal output:** Our system currently only predicts the steering angle of the car. Longitudinal decisions regarding when and how to stop the vehicle are solely based on hard-coded object detection rules.

V. Outlook

In this section, we outline the next steps and future directions for improving our autonomous driving system.

A. Next Steps and Future Directions

To enhance the performance and capabilities of our autonomous driving system, several avenues for improvement can be explored:

1) **Data and Pre-processing Enhancement:** We plan to improve the quality and diversity of our training data. Incorporating side images and employing better data balancing techniques will contribute to a more robust and adaptable model. This step aims to ensure that our system can handle a wide range of real-world scenarios effectively. Furthermore, collecting more data is a top priority once the simulator gets to a better level.
2) **Branched-Conditional Model:** Consider transitioning to a branched conditional model architecture instead of a concatenation-based architecture. This adjustment will offer improved modeling capabilities, enabling our system to make more informed and context-aware driving decisions. It will also facilitate smoother transitions between different driving scenarios.
3) **Enhanced Object Detection Rules:** We intend to refine the object detection-based braking rules. Currently the braking mechanism will sometimes brake when it is not necessary. By incorporating the size of bounding boxes, our system will be better equipped to assess potential obstacles accurately. This enhancement will contribute to safer and more efficient navigation, particularly in situations involving other vehicles.
4) **Instance Segmentation:** We are exploring the adoption of instance segmentation techniques instead of traditional object detection methods. Instance segmentation provides a more detailed and precise understanding of the environment, which translates into richer and more context-aware decision-making. This could help with navigating complex and cluttered scenes. Furthermore, an instance segmented map could be provided as an additional model input or replace the original input altogether.
5) **Speed as an Output:** We plan to add the car's speed as an additional output of our model. This inclusion will enable our system to have better control over its velocity, ensuring smoother acceleration and braking. Of course, true longitudinal navigation is a requirement for a truly self-driving car, and our current constant speed rules will not provide realistic driving capabilities in the real world.
6) **Real-time Predictions and Latency Reduction:** Currently, the simulator has some performance issues. Addressing challenges related to real-time predictions and simulator latency is a top priority. The current performance issues make it difficult to test our methods and to properly iterate the methodology.

References

[1] Bojarski, Mariusz, et al. End to End Learning for Self-Driving Cars. arXiv, 25 Apr. 2016. arXiv.org, https://doi.org/10.48550/arXiv.1604.07316.

[2] Codevilla, Felipe, et al. "End-to-end driving via conditional imitation learning." 2018 IEEE international conference on robotics and automation (ICRA). IEEE, 2018.

[3] Codevilla, Felipe, et al. "Exploring the limitations of behavior cloning for autonomous driving." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019.

[4] Codevilla, Felipe, et al. "On offline evaluation of vision-based driving models." Proceedings of the European Conference on Computer Vision (ECCV). 2018.

[5] Shao, Hao, et al. "ReasonNet: End-to-End Driving with Temporal and Global Reasoning." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2023.

[6] Wu, Penghao, et al. "Trajectory-guided control prediction for end-to-end autonomous driving: A simple yet strong baseline." Advances in Neural Information Processing Systems 35 (2022): 6119-6132.

[7] He, Kaiming, et al. Deep Residual Learning for Image Recognition. arXiv, 10 Dec. 2015. arXiv.org, https://doi.org/10.48550/arXiv.1512.03385.

[8] Jocher, G. (2020). YOLOv5 by Ultralytics (Version 7.0) [Computer software]. https://doi.org/10.5281/zenodo.3908559

[9] J. L. Binangkit and D. H. Widyantoro, "Increasing accuracy of traffic light color detection and recognition using machine learning," 2016 10th International Conference on Telecommunication Systems Services and Applications (TSSA), Denpasar, Indonesia, 2016, pp. 1-5, doi: 10.1109/TSSA.2016.7871074.

[10] Liu, Fei, Zihao Lu, and Xianke Lin. "Vision-Based Environmental Perception for Autonomous Driving." arXiv preprint arXiv:2212.11453 (2022).

[11] Hafiz, Abdul Mueed, and Ghulam Mohiuddin Bhat. 'A Survey on Instance Segmentation: State of the Art'. International Journal of Multimedia Information Retrieval, vol. 9, no. 3, Sept. 2020, pp. 171–89. arXiv.org, https://doi.org/10.1007/s13735-020-00195-x.