Modular System for Autonomous Driving

F. Basara, F. Baturan, I. Tica, A. Varga

Faculty of Technical Sciences, Novi Sad

Problem and motivation

Our main goal is to build a system for autonomous driving. Initial idea was to implement a system that would be trained in the CARLA simulator and then transferred to real world.

For that reason we chose to design a modular system that would allows us to have a sufficient level of abstraction between core unit and input data. Using this architecture we would be able to train control unit separately from task specific modules.

We identified four modules we believe are crucial for traffic-scene understanding. These are:

- Road segmentation,
- Traffic light detection,
- Pedestrian detection and
- Vehicle detection

Every module returns general information about the aspect of environment which it is in charge of. In this way, core module deals with just most important information about specific things around it. This concept helps core module to focus on important things from its environment.

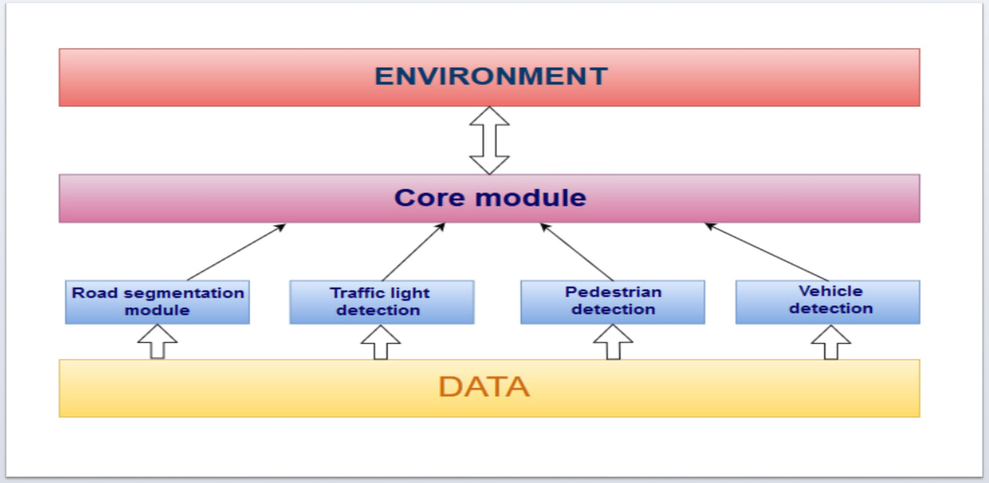
CARLA simulator

CARLA is an open-source **simulator** for autonomous driving research. CARLA consists mainly of two modules, the **CARLA Simulator** and the **CARLA Python API** module. The simulator does most of the heavy work, controls the logic, physics, and rendering of all the actors and sensors in the scene.

The CARLA provides a Python API that can be imported into Python scripts and it also provides an interface for controlling the simulator and retrieving data. Python API allows, for instance, control vehicles in the simulation, attach sensors to it, and read back the data these sensors generate.

Core module

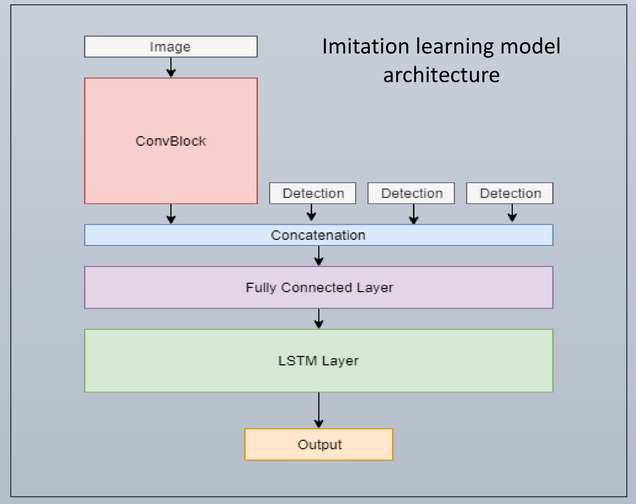
Core module architecture:



General pipeline begins with the frame provided from environment. That frame is then fed to each of the modules. Road segmentation module provides a segmented image of the road, while object detection modules provide positions and classes of various objects. Using this information core module generates controls (steer, throttle, break). This process is repeated for each frame.

The main advantage of this architecture is its independence from the number of modules and their specific implementation. Also the implementation of the core model can be based on different agents, such as imitation learning, reinforcement learning, evolutionary algorithms, etc.

Because of time constraints and hardware capabilities, we went for the imitation learning approach. We concluded that reinforcement learning performs significantly worse in the CARLA simulator while being much more compute intensive. It is worth mentioning that the training is done in supervised manner.



Data collecting and preprocessing

We acquired our data from CARLA simulator using its built-in autopilot mode. Data consist of images collected from two different cameras: the first one being RGB and the second one providing semantically segmented image. As every image goes through the same module prediction for every single epoch, which is very time-consuming, we decided to preprocess them in order to reduce training time.

Results

Results are much better than we initially anticipated. Unexpectedly our model performed well with very abstract and diverse data. Our agent succeeded in completing most of tasks, for example, taking a turn, stopping at red traffic light, stopping in front of car to avoid a collision…

However there are a lot of things that can be improved to make our agent drive event more smoother and precise.

Our main problem is agent's poor performance on crossroads. More advanced methods are required in order to achieve better performance in that segment of the environment. This problem is mostly caused because of our dataset: we recorded autopilot driving randomly around the city in the simulator, acquiring images with random actions: for example on crossroad A, our agent turns left in 50% of cases and turns right in 50% so it cannot learn where to go on that specific crossroad.

References

- CARLA simulator (http://carla.org/)
- YOLO (https://pjreddie.com/darknet/yolo/)
- YOLO Keras implementation (https://github.com/experiencor/keras-yolo2)
- Segmentation model implementation (https://github.com/qubvel/segmentation_models)
- CARLA paper (http://proceedings.mlr.press/v78/dosovitskiy17a/dosovitskiy17a.pdf)

Modular System for Autonomous Driving

F. Basara, F. Baturan, I. Tica, A. Varga

Faculty of Technical Sciences, Novi Sad

Vehicle detection module

Main goal for this module is to detect vehicles from CARLA simulator in real time. After detecting cars from the given image, the module returns vector of numbers, which carry information about car position on the image (dimensions and coordinates of the bounding boxes) as well as the confidence of detected object.

The model we used to implement this module is Tiny-YOLO. That is smaller (more compact) version of *You only look once* (*YOLO*), a state-of-the-art object detection system, which works in real-time. We used modified pre-trained model, trained on images from *COCO* (Common Objects in Context) dataset. Beside adding object detection layer to original Tiny-YOLO model, we froze first five layers and trained remaining ones on our dataset.

The dataset we used for training consists of more than 200 images from CARLA simulator, with provided annotations. It includes various scenes of vehicles captured from different angles and distances.

The module has a mAP of 70% on test images of our dataset. Prediction take approximately 0.1 seconds and it performs well both on small and big vehicles.

The precision could be improved with additional training on bigger dataset.

Traffic lights detection module

One of the most complex problems an autonomous vehicle is facing when driving in urban environments is deciding how to act on turns and crossroads. In that scenario a vehicle needs to have a complete traffic scene understanding, including certain predefined rules. Crucial for yielding an optimal solution is knowing the state and the location of the traffic light.

The idea behind this module is to predict in real time and with high confidence. That was accomplished using the Tiny YOLOv2 fully convolutional object detection architecture. When integrated into the core module, it offers information about the state and the location of the traffic light.

Model achieved a 66 mAP, 59 for the green light and 73 for the red light.

The model was first trained on around 5800 real world images from the LISA traffic light dataset.

After achieving good results on the LISA dataset, model was trained on around 1800 images collected by myself from the CARLA simulator.

Both datasets were slightly biased towards red lights.

- Bigger grid size is more suited for detecting small objects, so it would probably yield bigger accuracy
- Unfortunately it was hard to train since pre-trained weights weren't available for custom grid sizes
- Using tiny YOLOv3 would probably improve the model, since it has two predictions with grids of sizes 13 and 26
- Very difficult to train a stable feature extractor, required a lot of experimentation with loss scales, layer freezing and learning rates

Road segmentation module

Vehicle position and orientation on the road is an important information for driving. As we decided to use regular camera as the only source of data for our system, we had to come up with a solution of separating drivable paths and road lines from other objects on image.

Semantic image segmentation models were chosen as a solution for this problem as they are able to, using per pixel classification, produce new images that contain only desired information while preserving object structure. We explored four models with two different types of architectures, PSPNet and FPN that are using pyramidal feature pooling and encoder-decoder based UNet and LinkNet.

Dataset used for this module is compiled from images recorded during driving in CARLA simulator using built in autopilot. CARLA simulator also provides annotated target images for semantic segmentation. During batch generation, we covert every pixel of these annotated images to one-hot encoded vectors for simpler loss calculation. Semantic classes we are focusing on are road and road line.

Pedestrian detection module

The motivation behind "pedestrian detection module" is to be able to detect any potential collision of a vehicle with the pedestrians on the road. The crucial task of this module is to find a rectangle that best represents a person that is in front of a vehicle.

Dataset that was used for its training was made by annotating every person in 300 images made in Carla simulator. Data set includes different poses of pedestrians, on the different scenes on the road, as well as vast amount of distances between a car and a person.

Detection algorithm that was used was "yolo" and it was fine tuned to adapt to the new data. "Tiny yolo" configuration did not give results good enough to be used for driving a car in a simulator. For that reason, full yolo configuration was used and the results were much better as feature extractor part of the yolo algorithm .

Most of the time, the module is capable of detecting a person, but the bounding box that represents it does not always have the best possible coordinates. When tested on pedestrians that are close to the car, prediction works much better than when they are far away, which we think is good enough result for our specific problem. Still, model is often unable to detect when two persons walk next to each other or when only a part of a body is visible from the car camera. Also, sometimes a detection rectangle can be put on an object that is not a person, especially if that object is narrow or far away and that could represent a problem while driving, but while vehicle is getting closer to the object, model can dismiss a prediction and continue to drive.