L8 Autonomous Driving with IL&RL

Zonghua Gu 2022

Outline

- Introduction to IL and IRL
- Paper discussions

RL Reward Function Issues

- RL can be very effective, provided that a suitable reward function is available.
- Bad reward function leads to undesirable behavior
  - Suppose you design a vacuum cleaner to maximize reward function defined as “cumulative amount of dirt sucked in”. The vacuum cleaner may learn to repeatedly spit out and suck in the same pile of dirt!
- For AD in realistic environment, the goal is to reach destination with minimum time, while avoiding accidents. But how to encode this into a reward function?
  - “A great reward function can help you better optimize your reinforcement learning model. In AWS DeepRacer, the reward function is written in Python code, and uses different input parameters to help encourage good behavior and disincentivize poor behavior. There’s no single right answer for which parameters to include in your reward function, and your best reward function will likely require a lot of experimentation.” from AWS DeepRacer MOOC.

Imitation Learning (IL)

- IL is useful when it is easier for an expert to demonstrate the desired behavior, rather than to specify a reward function for RL to learn the policy.
  - e.g., for Automated Driving in a realistic environment, the reward function would be a huge complex function involving everything in the environment
  - Requires access to an expert, either offline (driving data logs) or online (query-on-demand)
  - Also called Learning from Demonstrations.
- IL variants:
  - Learn to mimic the expert's policy
    - Behavior Cloning
    - Direct Policy Learning
  - Learn the expert's value function
    - Inverse Reinforced Learning

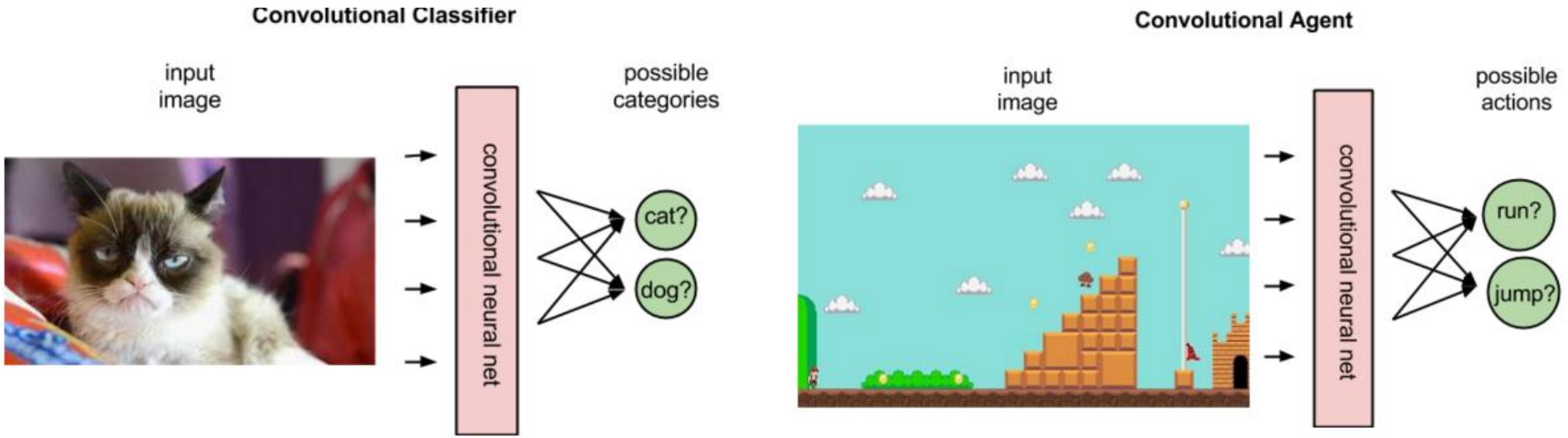
Notations

- State: s (sometimes x)
- Action: a (sometimes y)
- Policy π_θ (sometimes h)
  - Deterministic policy $a = \pi_\theta(s)$
  - Stochastic policy $P(a) = \pi_\theta(s)$
- Environment Model: $P(s'|s, a)$
  - Known: model-based
  - Unknown: model-free
- Rollout: sequentially execute policy $\pi_\theta(s_0)$ from initial state s_0 until timestep T.
  - Produce trajectory $\boldsymbol{\tau} = (s_0, a_0, s_1, a_1, \dots)$

Behavior Cloning (BC)

- Behavior Cloning (BC) is a form of Supervised Learning (SL), where an agent is trained to perform a task from demonstrations by learning a mapping (e.g., a CNN) between states (input data) and actions (labels).

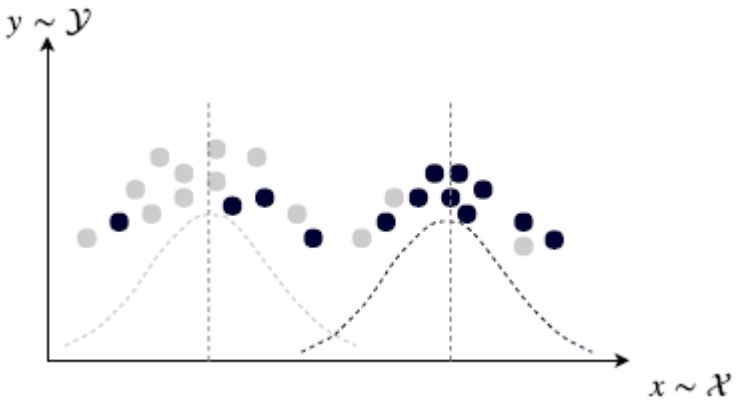
1. Collect demonstrations (τ^* trajectories) from expert
2. Treat the demonstrations as i.i.d. state-action pairs: $(s_0^*, a_0^*), (s_1^*, a_1^*), \ldots$
3. Learn π_θ policy using supervised learning by minimizing the loss function $L\left(a^*, \pi_\theta(s)\right)$



https://pathmind.com/wiki/deep-reinforcement-learning

SL vs. BC

- Main difference between SL and BC:
  - For SL: input x for computing label $y = F(x)$ is i.i.d (independent and identically distributed)
    - i.e., there is no correlation between one input image x_i and the next one x_{i+1}.
  - For IL/BC (and MDP in general): input (state) s_t for computing action $a_t = \pi_\theta(s_t)$ is not i.i.d. but highly correlated, since action taken in a given state s_t induces the next state s_{t+1}.
    - i.e., a vehicle does not randomly jump around, but follows a smooth path. If state s_t is defined as the front-camera video frame at time t, then there is strong correlation between frames across time s_t, s_{t+1} ... in the continuous video stream.
- Non-i.i.d input data may cause distributional Shift, where training and testing input data distributions densities are different.



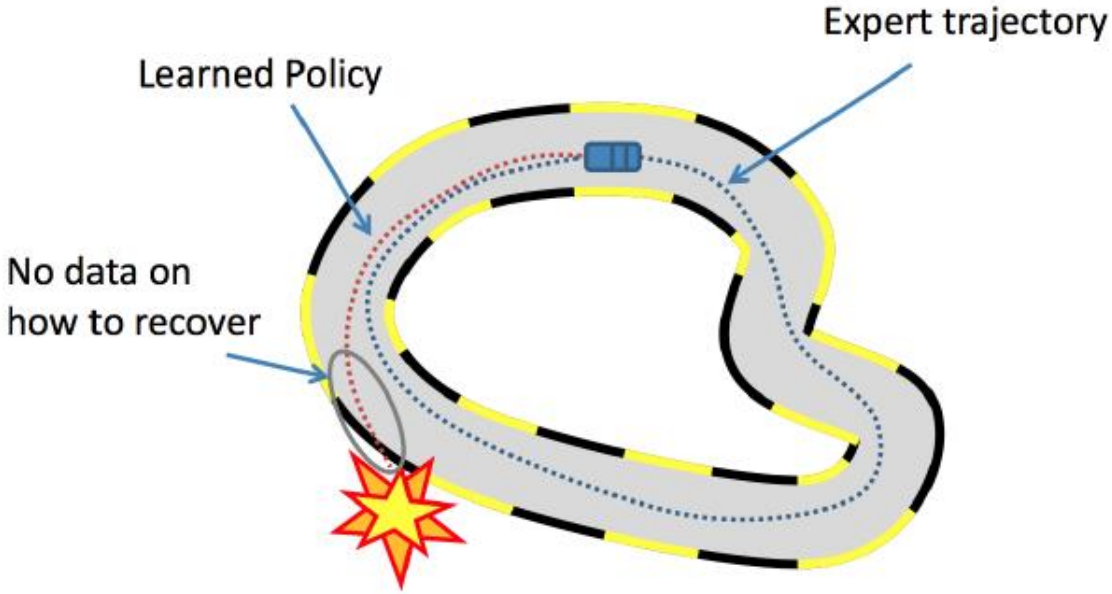
https://pathmind.com/wiki/deep-reinforcement-learning

Early Projects of IL (BC) Applied to AD

- ALVINN
  - CMU, 1990
  - Low-res image as input
  - Fully connected NN
- DAVE
  - Muller, LeCun, 2003
  - Low-res image as input
  - CNN

Distributional Shift in IL/BC

- Errors made in different states add up, therefore a mistake made by the agent can easily put it into a state that the expert has never visited and the agent has never trained on. In such states, the action is undefined and this can lead to catastrophic failures.
- e.g. the expert driver always keeps in the center of lane, so the front camera images (input data) in the training set do not contain views where the vehicle is heading to go off side of road. So once the vehicle heading deviates a little towards the side
  - the input data is not in the training set → action is undefined → vehicle deviates even more → ⋯ → vicious cycle leading to a crash.



https://medium.com/@SmartLabAI/a-brief-overview-of-imitation-learning-8a8a75c44a9c

Inverse Reinforcement Learning (IRL)

- Start with a set of expert's demonstrations (we assume these are optimal) and then try to estimate the parameterized reward function, that would cause the expert's behavior/policy.
  - Problem: reward function is not unique, multiple reward functions may lead to the same behavior.
  - e.g., an outside observer sees that you work very hard (behavior). He may infer your reward function to be "*maximize WorkTime*" (since you really enjoy your job). But your actual reward function is "*maximize MoneyEarned*" (while you really hate your job).
  - An agent with incorrect value function won't generalize well in a different environment. Suppose someone trains an agent to imitate you. The agent has the same behavior as you in the current environment, but faced with a job offer with higher salary, the agent may take a different action than you.
  - Solution: try to learn the correct value function by observing the expert in diverse environments.

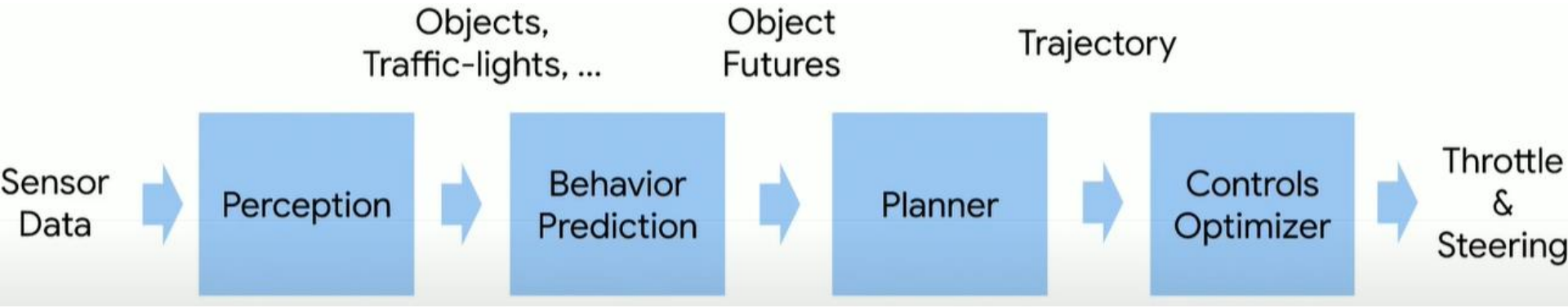
IRL Details

- Repeat until we find a good enough policy:
  - Update the reward function parameters.
  - Solve the RL problem to find the optimal policy.
  - Compare the newly learned policy with the expert's policy.

- Collect expert demonstrations: $D = \{\tau_1, \tau_2, \dots, \tau_m\}$
- In a loop:
  - Learn reward function: $r_\theta(s_t, a_t)$
  - Given the reward function r_θ, learn π policy using RL
  - Compare π with π^* (expert's policy)
  - STOP if π is satisfactory

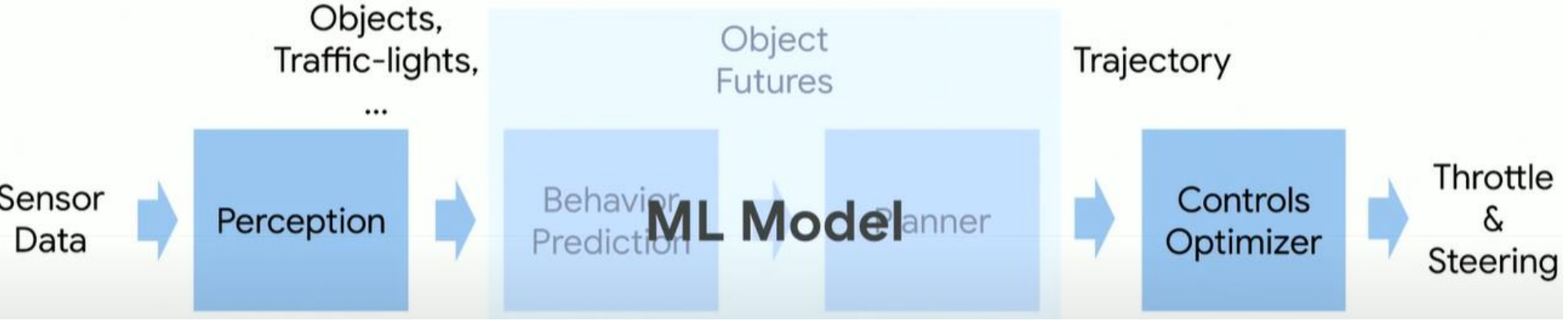
Outline

- Introduction to IL and IRL
- Paper discussions



Classic Pipeline

End-to-End Driving (NVIDIA's PilotNet)



Mid-to-Mid Driving (Waymo's ChauffeurNet)

Advantages of Mid-to-Mid

- On the left:
  - End-to-end: Raw sensor data contains extremely high dimensional information which can be influenced by different textures and appearances of roads and objects, different weather conditions, and different daytime.
  - Mid-to-mid: Separate perception module, the bird-view representation is a concise description of only the useful information for decision making and planning, discarding irrelevant information such as texture, light conditions and object appearances
- On the right:
  - IL needs labeled data in the form of expert driver's action a_i^* at each step i to obtain a trace of (s_i^*, a_i^*).
  - End-to-end: must record expert driver's low-level actions (steering/brake/acceleration) by tapping into and capturing signals from the vehicle's internal bus.
  - Mid-to-mid: only need to record expert driver's vehicle trajectory logs.
- Overall:
  - Mid-to-mid: Execution frequency of ML components is lower than end-to-end, since planner typically executes at lower freq than controller.

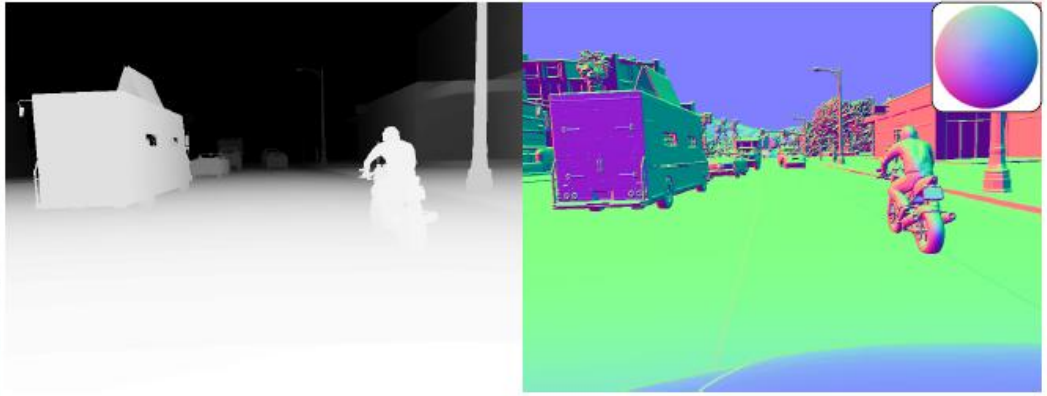
[Zhou 2019]

- Zhou, Brady, Philipp Krähenbühl, and Vladlen Koltun. "Does computer vision matter for action?." *arXiv preprint arXiv:1905.12887* (2019). (Intel Labs, UT Austin)

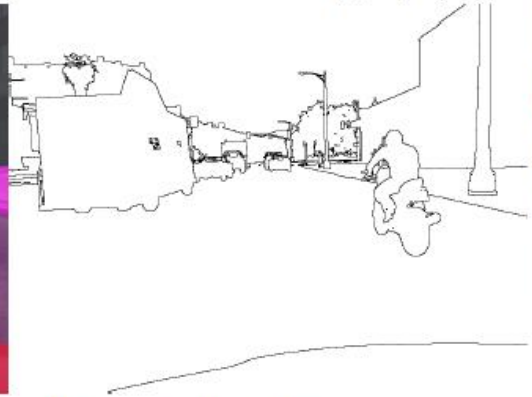
Computer Vision vs. End-to-End

- Computer vision tasks as perception module
  - Object recognition, depth estimation, optical flow, semantic segmentation…
  - e.g., train a model (CNN or others) with a large dataset to classify Stop Signs from images.
- End-to-end approach
  - Map input images (raw pixels) directly to action, bypassing explicit computer vision tasks. Perceptual capabilities will arise as needed, as a result of training for the specific task.
  - e.g., if the training dataset is highway driving, then E2E model will never learn features of Stop Signs;
  - If the training dataset is urbane driving, then E2E model may learn features of Stop Signs implicitly in the intermediate layers of a CNN, during the process of learning the mapping from “Stop Sign” input images to “braking” action. But there is no explicit classification layer that outputs the label “StopSign”.
- We report controlled experiments that assess whether specific vision capabilities are useful in mobile sensorimotor systems.
- Does Computer Vision Matter for Action?
  - https://www.youtube.com/watch?v=4MfWa2yZ0Jc

(a) RGB Image



(b) Depth (left) and surface normals (right)



(c) Segmentation: semantic (left) and instance boundaries (right).

(d) Albedo

(e) Optical Flow. Full (left) and factored into static (center) and dynamic flow (right).

Fig. S7. Different computer vision modalities used in our experiments, illustrated on the urban driving task. For normal maps, the inset shows the different normal directions projected onto a virtual sphere. For optical flow, the inset shows the flow direction as an offset to the center pixel.

Performance Evaluation Results

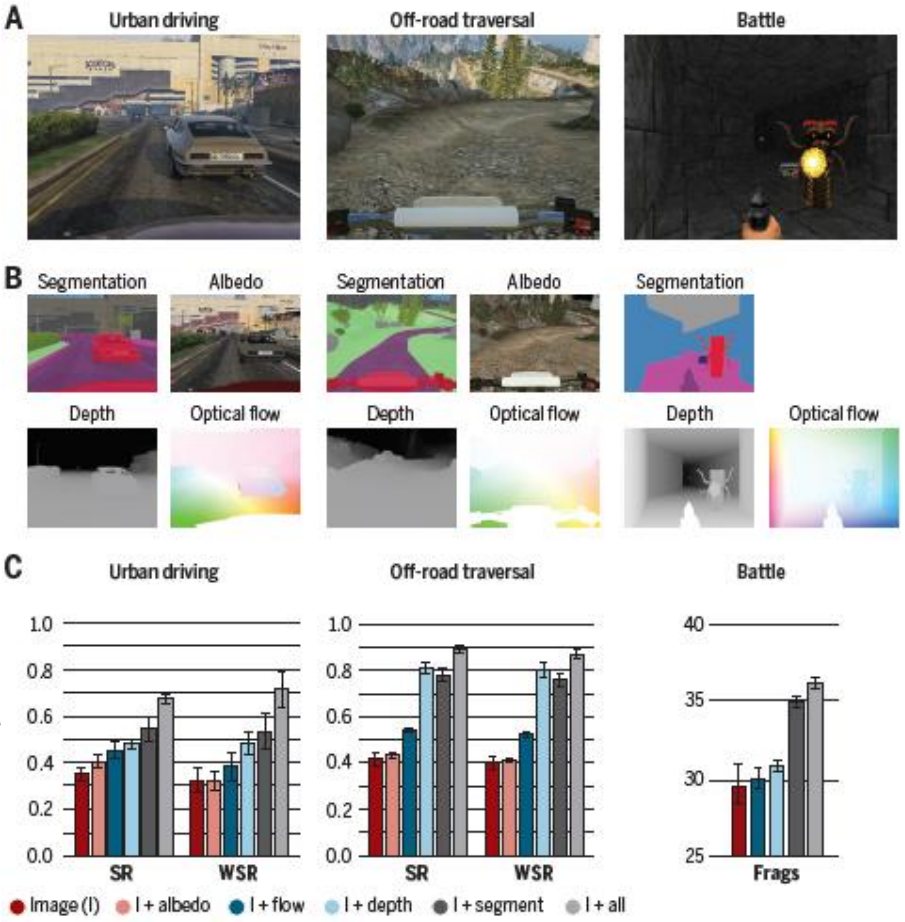


Fig. 1. Assessing the utility of intermediate representations for sensorimotor control. (**A**) Sensorimotor tasks. From left to right: urban driving, off-road trail traversal, and battle. (**B**) Intermediate representations. Clockwise from top left: semantic segmentation, intrinsic surface color (albedo), optical flow, and depth. (Albedo not used in battle.) (**C**) Main results. For each task, we compare an image-only agent with an agent that is also provided with ground-truth intermediate representations. The agent observes the intermediate representations during both training and testing. Success rate ('SR') is the fraction of scenarios in which the agent successfully reached the target location; weighted success rate ('WSR') is weighted by track length; 'frags' is the number of enemies killed in a battle episode. We show mean and standard deviation in each con-

Conclusions

- Computer vision does matter.
- When agents are provided with representations studied in computer vision, they achieve higher performance in sensorimotor tasks.
- Some computer vision capabilities appear to be more impactful for mobile sensorimotor operation than others. Specifically, depth estimation and semantic segmentation provide the highest boost in task performance.
- My thoughts: this explains rising popularity of mid-to-mid over end-to-end approaches.

PilotNet [Bojarski 2016]

- Bojarski M, Del Testa D, Dworakowski D, et al. End to end learning for self-driving cars[J]. arXiv preprint arXiv:1604.07316, 2016.

End-to-End Driving (NVIDIA's PilotNet)

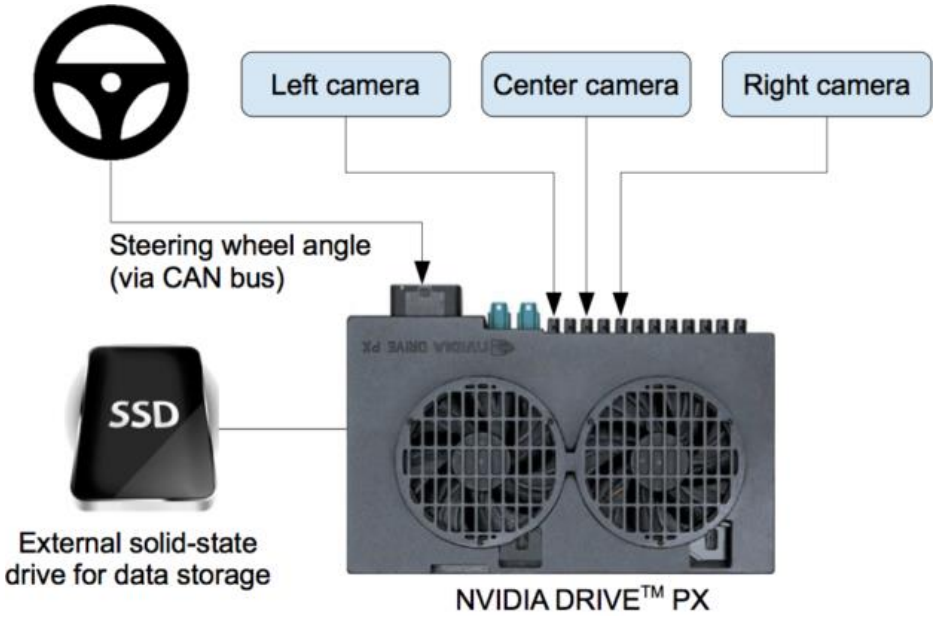
Training Method

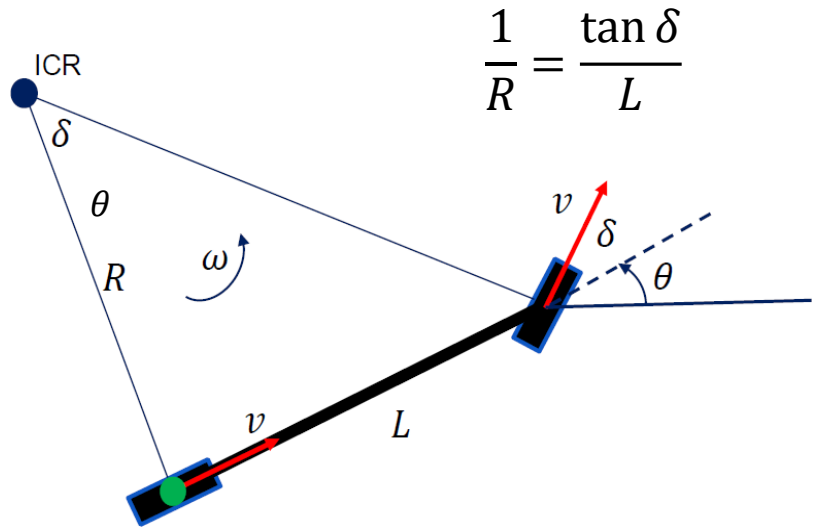
- End-to-End model trained with Imitation Learning (BC)
  - Human drives vehicle
  - Record sensor data and human actuator commands as training pairs
  - Train a DNN to map sensor data to actuator commands, mimicking a human. (PilotNet controls steering only. I think acceleration/braking are controlled separately, but the paper did not say how.)
- PilotNet driving video:
  - https://www.youtube.com/watch?v=_N7nC-8YxzE

https://github.com/lhzlhz/PilotNet

Data Collection System

- Three cameras are mounted behind the windshield of the data-acquisition car, and timestamped video from the cameras is captured simultaneously with the steering angle applied by the human driver. The steering command is obtained by tapping into the vehicle's Controller Area Network (CAN) bus.
- In order to make our system independent of the car geometry, we represent the steering command as $\frac{1}{R}$, where R is the turning radius in meters. We use $\frac{1}{R}$ instead of R to prevent a singularity when driving straight (the turning radius for driving straight is infinity). $\frac{1}{R}$ smoothly transitions through zero from left turns (negative values) to right turns (positive values).
- Training data contains single images sampled from the video, paired with the corresponding steering command ($\frac{1}{R}$).

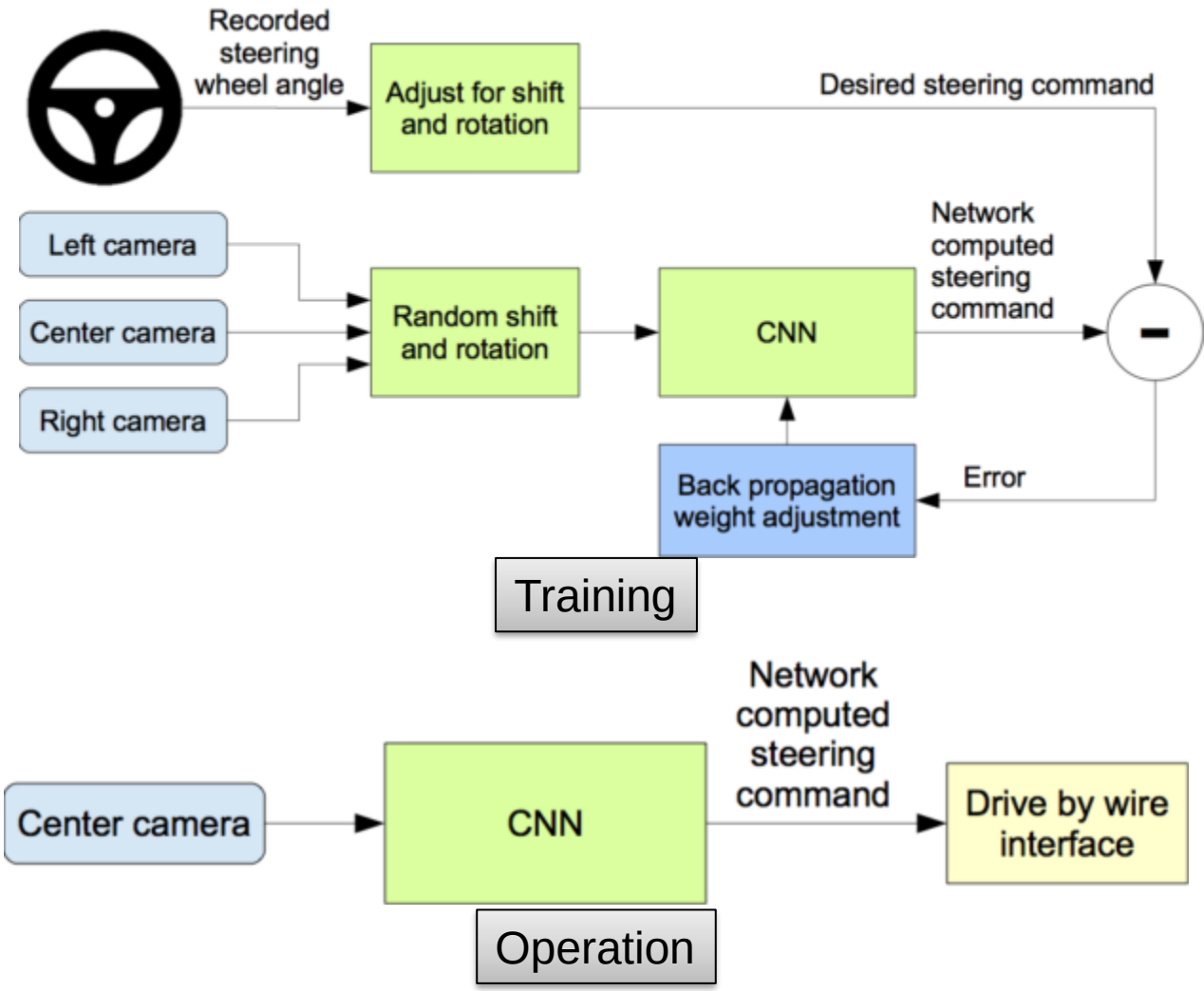




https://devblogs.nvidia.com/deep-learning-self-driving-cars/

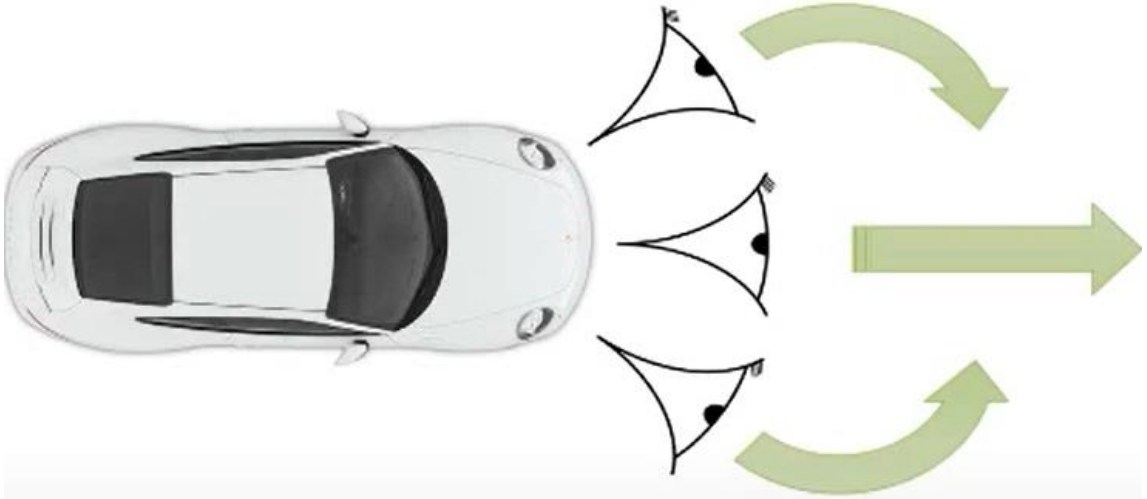
Training Data Augmentation

- Problem: training data from human driver always stays in center of lane. So if car deviates from center, the image it sees is not in the training set, so it does not know the correct action.
- Solution: Augment training data with additional images that show the car in different shifts from the center of the lane and rotations from the direction of the road. The images for two specific off-center shifts can be obtained from the left and the right cameras.



Training Data Augmentation Details

- We augment the data by adding artificial shifts and rotations to teach the network how to recover from a poor position or orientation.
- Images for two specific off-center shifts can be obtained from the left and the right camera. Additional shifts between the cameras and all rotations are simulated by viewpoint transformation of the image from the nearest camera. The steering label for transformed images is adjusted to one that would steer the vehicle back to the desired location and orientation in two seconds.
- Ex.: Image from center camera shows car is driving straight; Shifted image from left camera shows car is leaning left. Associate this image with synthetic “turn right” command (even though human driver never issued it). During operation, if the center camera sees this image, issue “turn right” command.



Driving Simulator

- The simulator transforms the original images to account for departures from the ground truth. The magnitude of these perturbations is chosen randomly from a normal distribution.

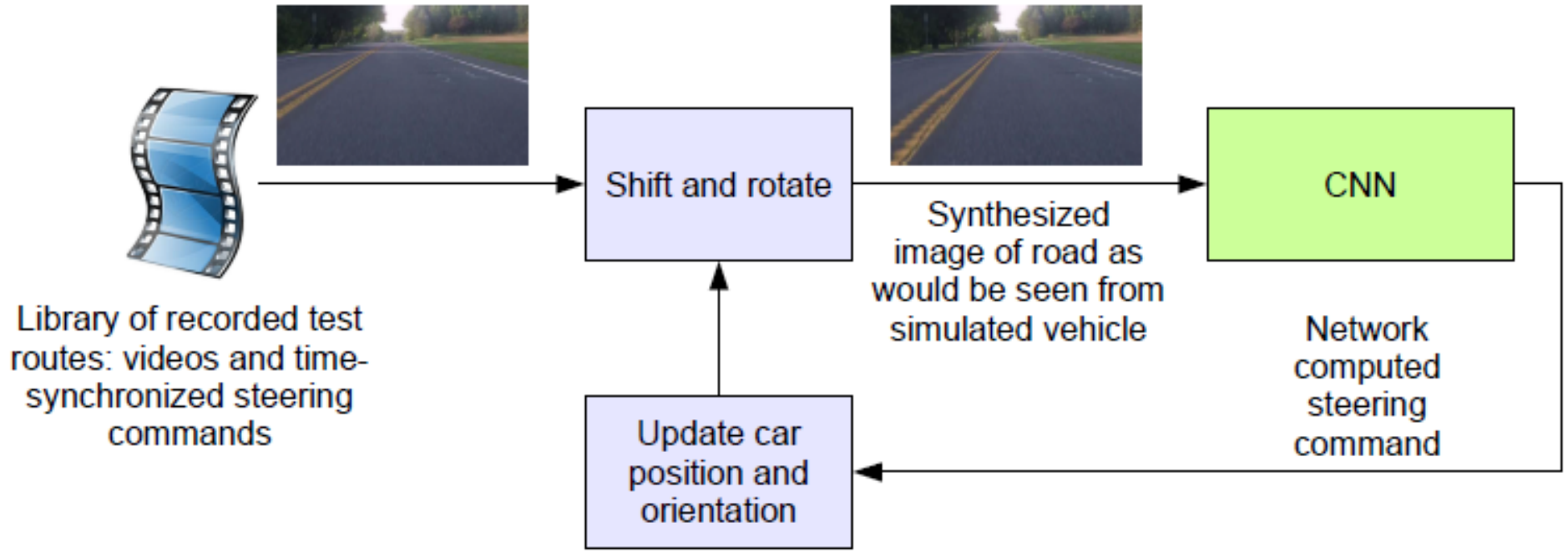


Figure 5: Block-diagram of the drive simulator.

PilotNet

- ~250K distinct weights
- ~27M connections

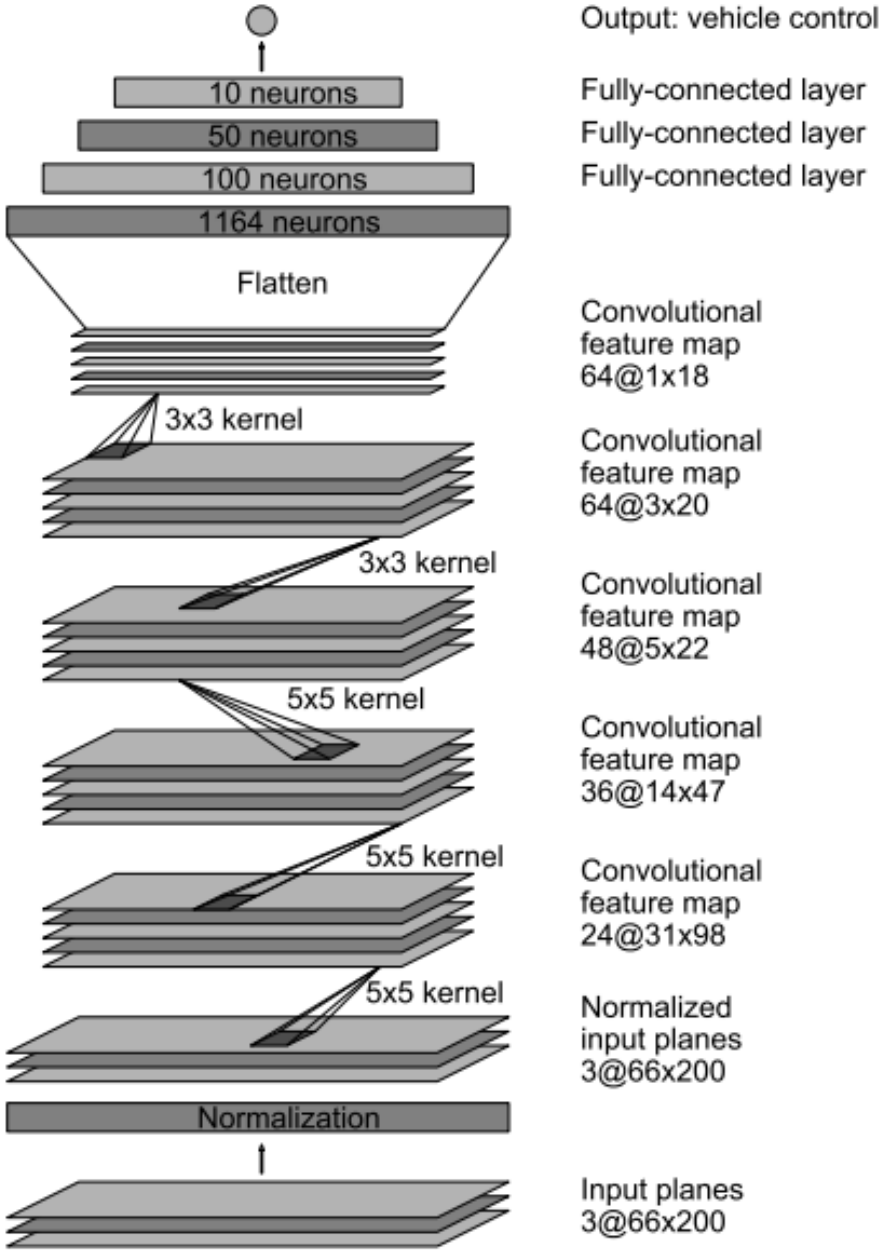


Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters. PilotNet

Visualization of Salient Objects

- The visualization shows which regions of the input image contribute most to the output of the network. These regions identify the salient objects (highlighted in green).
- PilotNet focuses on the same things a human driver would, including lane markers, road edges and other cars.

https://blogs.nvidia.com/blog/2017/04/27/how-nvidias-neural-net-makes-decisions/
Bojarski M, Yeres P, Choromanska A, et al. Explaining how a deep neural network trained with end-to-end learning steers a car[J]. arXiv preprint arXiv:1704.07911, 2017.

Not covered in exam

Visualization Method

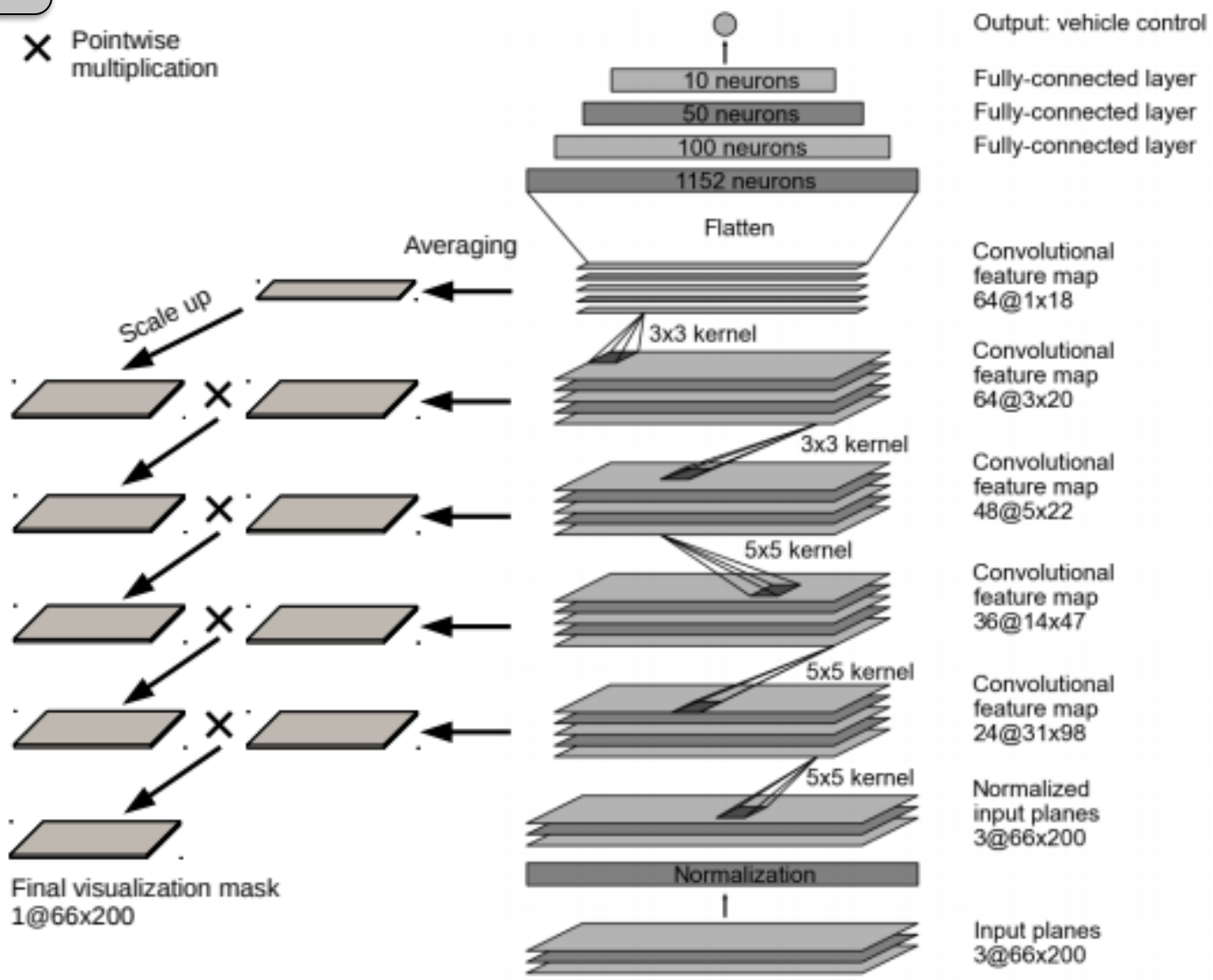


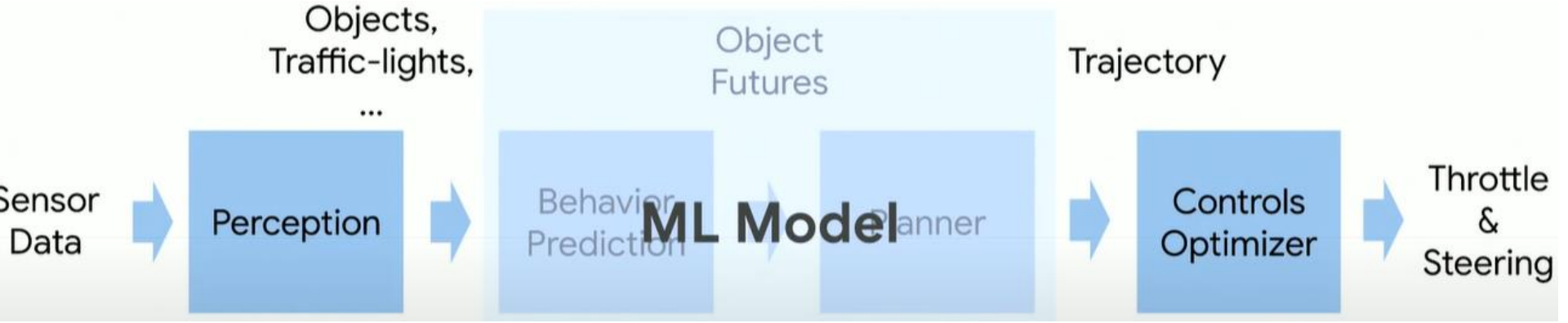
Figure 2: Block diagram of the visualization method that identifies the salient objects.

On-Road Tests

- After a trained network has demonstrated good performance in the simulator, the network is loaded on the DRIVE PX in our test car and taken out for a road test.
- For a typical drive in Monmouth County NJ from our office in Holmdel to Atlantic Highlands, we are autonomous approximately 98% of the time. We also drove 10 miles on the Garden State Parkway (a multi-lane divided highway with on and off ramps) with zero intercepts.

Chauffeurnet [Bansal 2018]

- Bansal M, Krizhevsky A, Ogale A. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst[J]. arXiv preprint arXiv:1812.03079, 2018.



Mid-to-Mid Driving (Waymo's ChauffeurNet)

https://medium.com/waymo/learning-to-drive-beyond-pure-imitation-465499f8bcb2

Abstract

- Our goal is to train a policy for autonomous driving via imitation learning that is robust enough to drive a real vehicle. We find that standard behavior cloning is insufficient for handling complex driving scenarios, even when we leverage a perception system for preprocessing the input and a controller for executing the output on the car: 30 million examples are still not enough.
  - Training data: 30 million real-world expert driving examples, corresponding to about 60 days of continual driving
- We propose exposing the learner to synthesized data in the form of perturbations to the expert's driving, which creates interesting situations such as collisions and/or going off the road.
- Rather than purely imitating all data, we augment the imitation loss with additional losses that penalize undesirable events and encourage progress – the perturbations then provide an important signal for these losses and lead to robustness of the learned model.

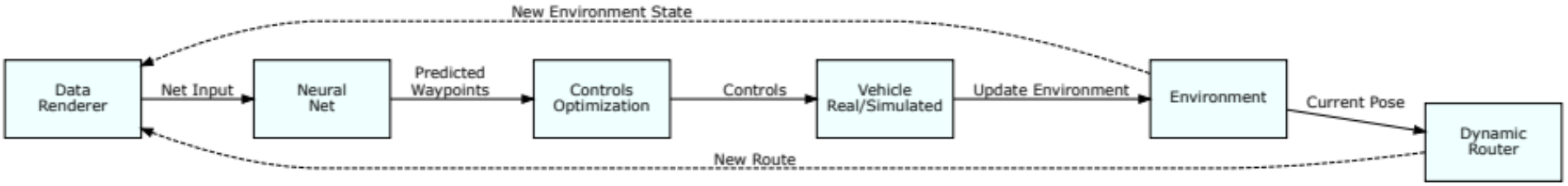
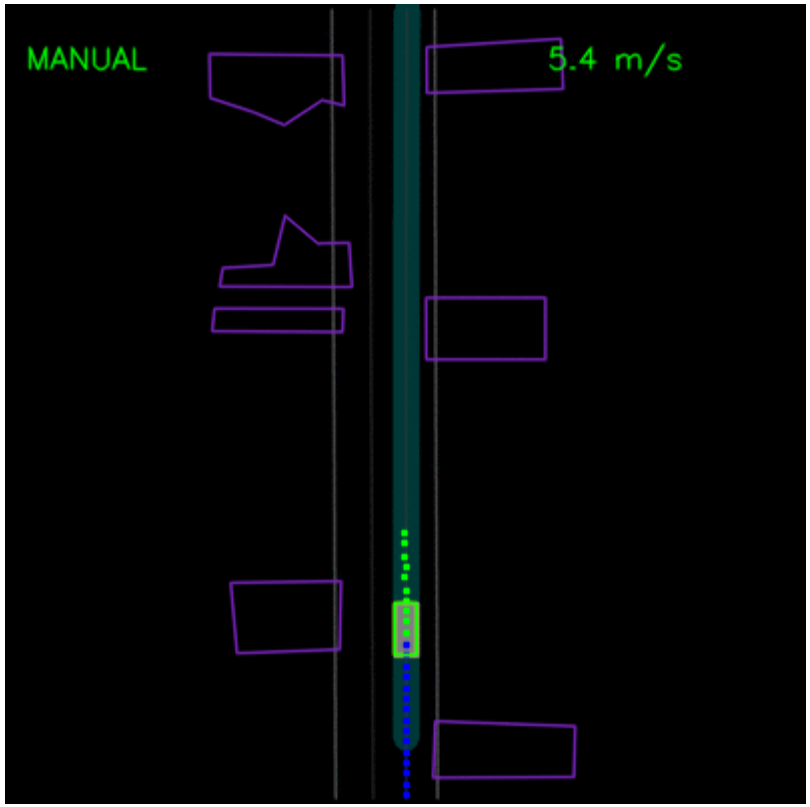


Figure 4: Software architecture for the end-to-end driving pipeline.

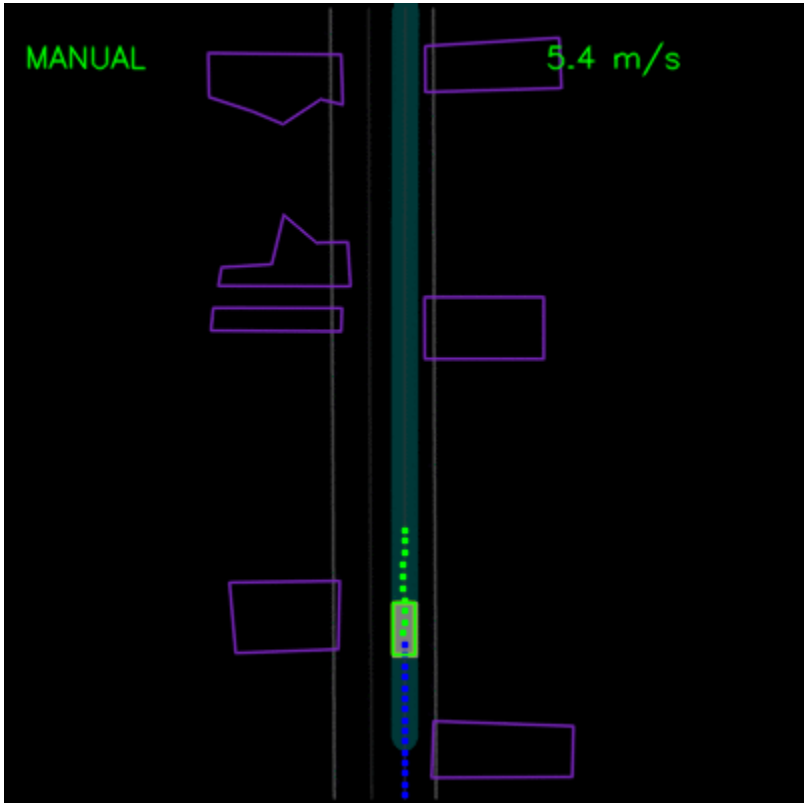
BC Experiments

- Agent trained with pure BC gets stuck behind a parked vehicle (left) and is unable to recover from a trajectory deviation while driving along a curved road (right). The teal path depicts the input route, yellow box is a dynamic object in the scene, green box is the agent, blue dots are the agent's past positions and green dots are the predicted future positions.



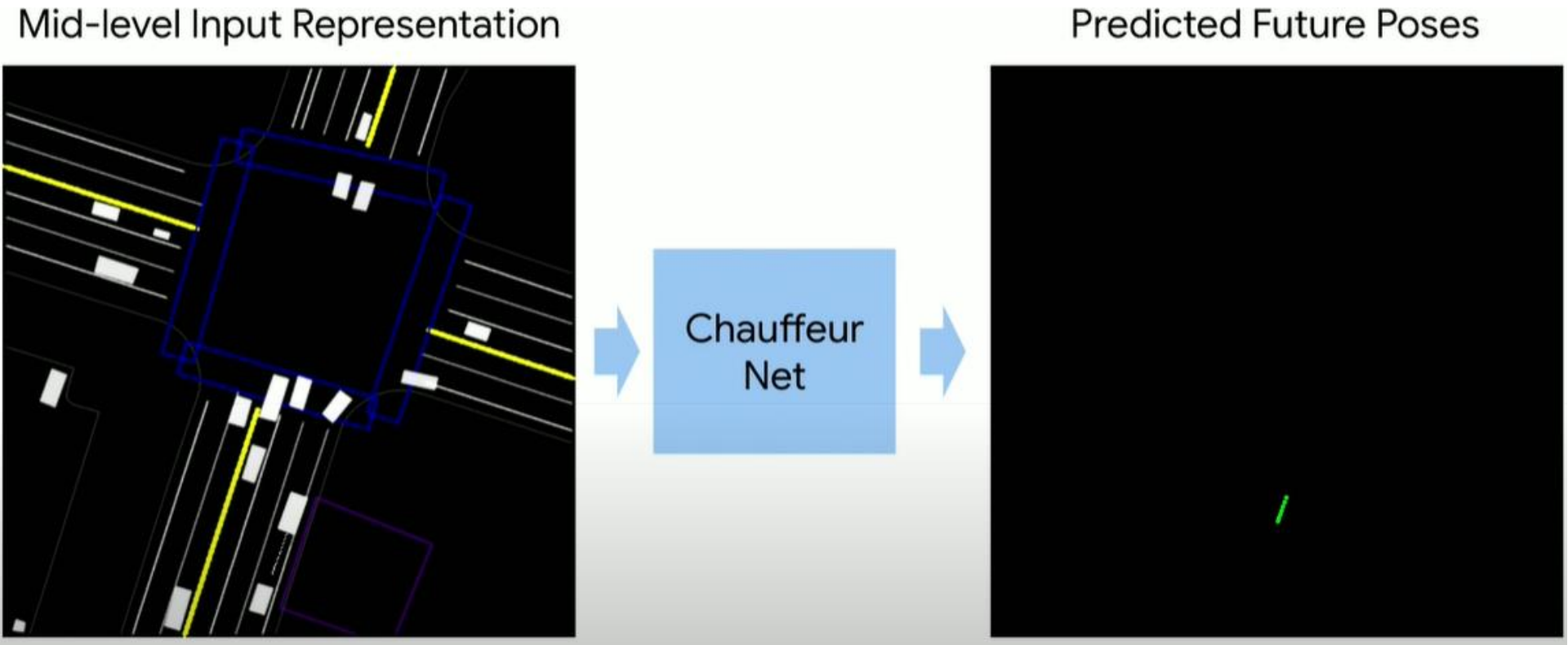
ChauffeurNet Experiments

- *ChauffeurNet* model can now successfully nudge around the parked vehicle (left) and recover from the trajectory deviation to continue smoothly along the curved road (right).

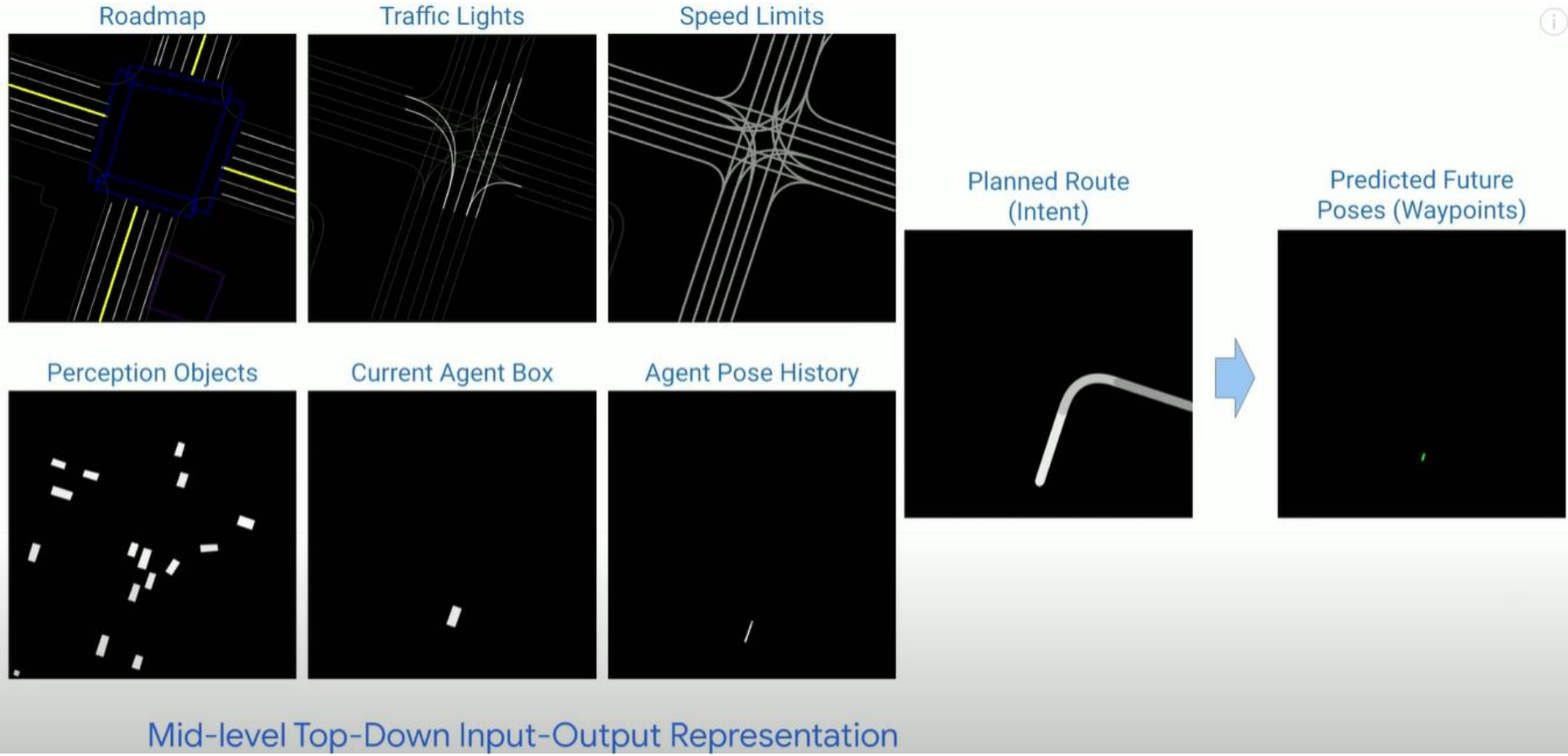


ChauffeurNet Input/Output

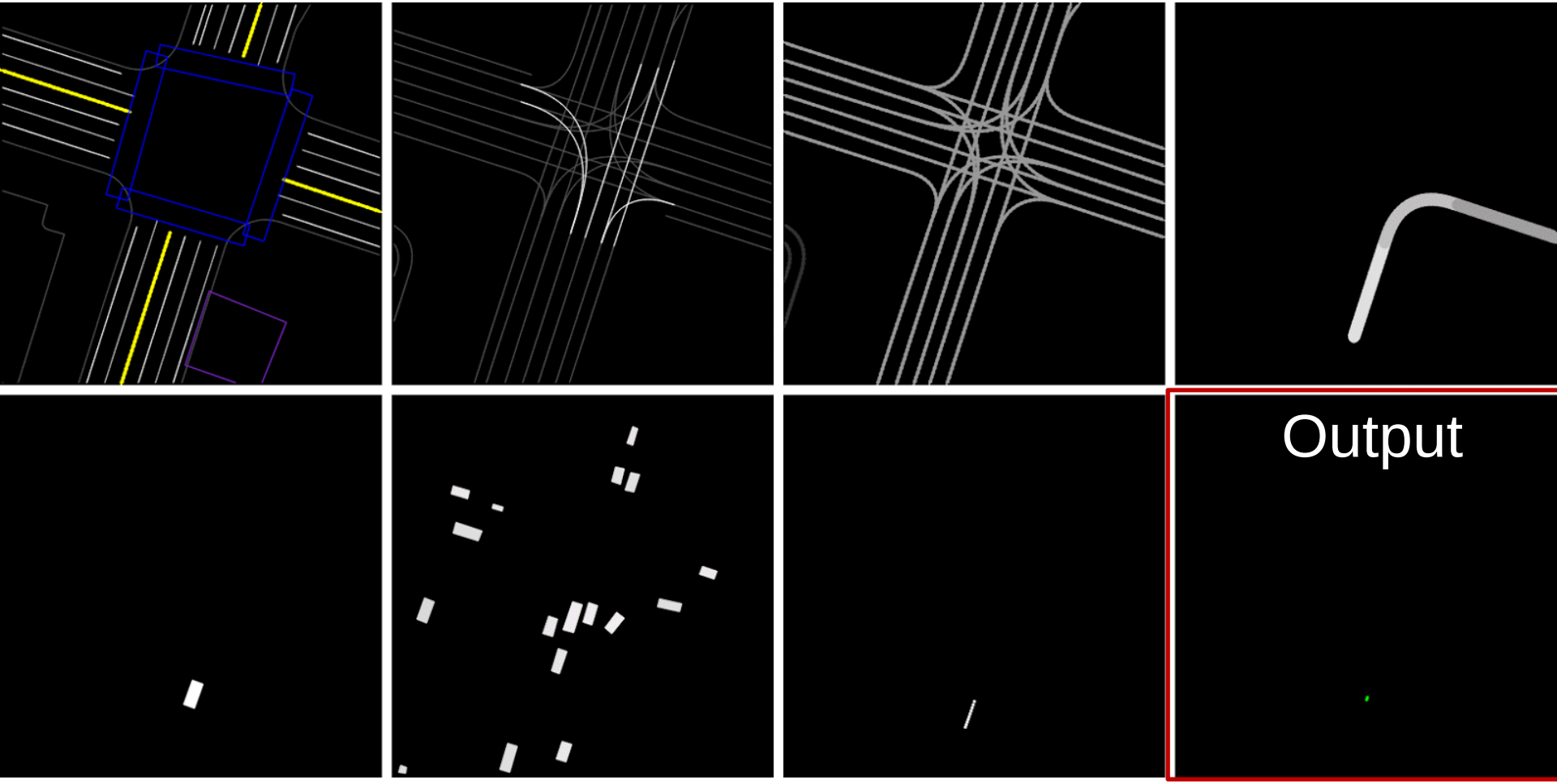
- We use a perception system that processes raw sensor information and produces our input: a top-down representation of the environment and intended route, where objects such as vehicles are drawn as oriented 2D boxes along with a rendering of the road information and traffic light states. We present this mid-level input to a recurrent neural network (RNN), named ChauffeurNet, which then outputs a driving trajectory that is consumed by a controller which translates it to steering and acceleration.



ChauffeurNet Input/Output in Detail



Rendered inputs and outputs



- **Top-row left-to-right:** Roadmap, Traffic lights (bright lines denote red light), Speed-limit, and Planned Route (intent: ego-car should turn right).
- **Bottom-row left-to-right:** Current Agent Box, Dynamic Boxes (history of past 1-2s), Past Agent Poses, and the output Future Agent Poses (shown in last slide).

ChauffeurNet Architecture

- ChauffeurNet is a Convolutional Recurrent Neural Network (RNN), consisting of the CNN FeatureNet, and the AgentRNN.
- Trained with IL, guided by ground truth data (green) and loss functions (blue)
  - 26M examples from real driving logs

Conditional IL (CIL) [Codevilla 2018]

- Codevilla, Felipe, et al. "End-to-end driving via conditional imitation learning." *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018.

Standard IL Limitations

- Standard IL: training data is a set of observation-action pairs $D\{\langle o_i, a_i \rangle\}_{i=1}^{N}$ generated by expert. We train a function (DNN) to mimic expert actions a_i
  - $\min_{\theta} \sum_i l(F(o_i; \theta), a_i)$ (Note different notations: here o_i is the same as state s_i we saw before, e.g., video captured by front camera.)
  - An implicit assumption of this formulation is that the expert's actions are fully explained by the observations; that is, there exists a function E that maps observations to the expert's actions: $a_i = E(o_i)$
- IL works well if this assumption holds, e.g., for lane following. However, in more complex scenarios the assumption breaks down. Consider a driver approaching an intersection. His subsequent actions are not explained by his observations, but are additionally affected by his intention (do I want to turn left, go straight or turn right?). The same observations could lead to different actions, depending on this intention.

Command Conditional IL

- At training time, command c is provide by expert, e. g., go (left, straight, right) a few seconds before reaching next intersection.
  - e.g., human drivers already use turn signals to communicate their intention when approaching intersections; these turn signals can be used as commands.
  - The training dataset becomes $D\{\langle o_i, c_i, a_i \rangle\}_{i=1}^{N}$. Command-conditional IL objective $\min_{\theta} \sum_i l(F(o_i, c_i ; \theta), a_i)$
- At test time, commands can come from a human user (passenger), or a high-level planning module (with A*, Dijkstra…). They affect behavior of the controller in two possible architectures.
  - (My comment: shouldn't the high-level planner always be present in any AD system?)
- ICRA 2018 Spotlight Video
  - https://www.youtube.com/watch?v=GNVHds_mvlg

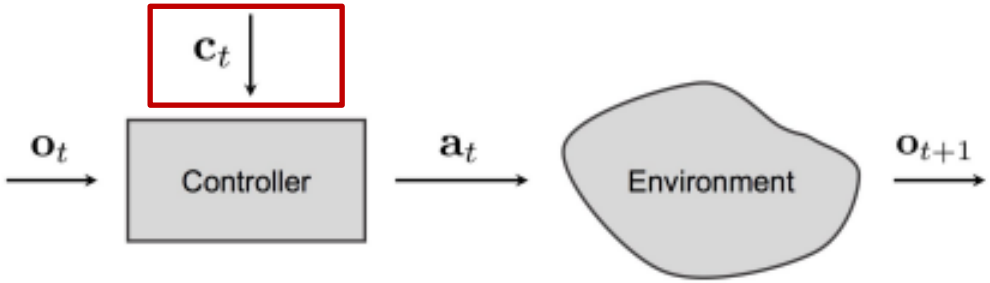


Fig. 2. High-level overview. The controller receives an observation $\mathbf{o}_t$ from the environment and a command $\mathbf{c}_t$. It produces an action $\mathbf{a}_t$ that affects the environment, advancing to the next time step.

Two Architectures

- Each observation $o = \langle i, m \rangle$ comprises
  - Front camera image i
  - and a low-dimensional vector m (e.g., car speed)
- Action $\boldsymbol{a} = \langle s, a \rangle$ (steering angle and acceleration)
- Loss function for each sample: $l(\boldsymbol{a}, \boldsymbol{a}_{gt}) = l(\langle s, a \rangle, \langle s_{gt}, a_{gt} \rangle) = \|s - s_{gt}\|^2 + \lambda_a \|a - a_{gt}\|^2$ (gt is ground truth from expert demo)
- Left: Command c is one of the inputs.
- Right: Command c acts as a switch.

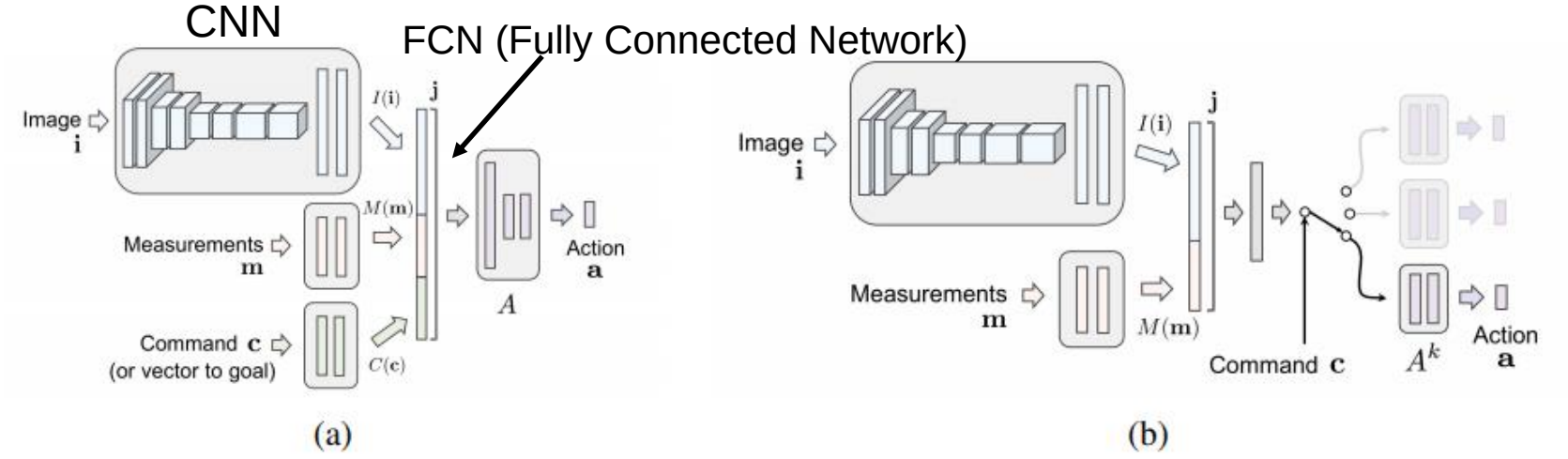


Fig. 3. Two network architectures for command-conditional imitation learning. (a) `command input`: the command is processed as input by the network, together with the image and the measurements. The same architecture can be used for goal-conditional learning (one of the baselines in our experiments), by replacing the command by a vector pointing to the goal. (b) `branched`: the command acts as a switch that selects between specialized sub-modules.

Data Augmentation

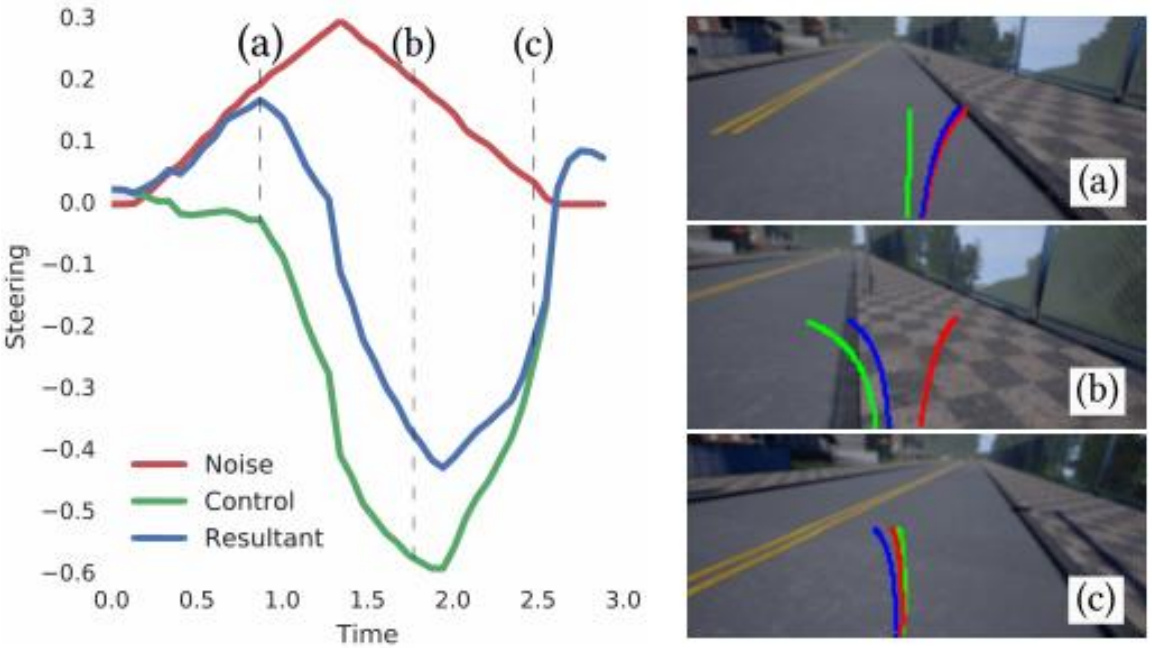


Fig. 4. Noise injection during data collection. We show a fragment from an actual driving sequence from the training set. The plot on the left shows steering control [rad] versus time [s]. In the plot, the red curve is an injected triangular noise signal, the green curve is the driver's steering signal, and the blue curve is the steering signal provided to the car, which is the sum of the driver's control and the noise. Images on the right show the driver's view at three points in time (trajectories overlaid post-hoc for visualization). Between times 0 and roughly 1.0, the noise produces a drift to the right, as illustrated in image (a). This triggers a human reaction, from 1.0 to 2.5 seconds, illustrated in (b). Finally, the car recovers from the disturbance, as shown in (c). Only the driver-provided signal (green curve on the left) is used for training.

Comparisons

- View representation:
  - PilotNet and CIL [Codevilla18] use front camera view
  - ChauffeurNet uses bird-view representation, which helps simplify the visual information while maintaining useful information for driving.
- Where DNN is used:
  - PilotNet and CIL are end-to-end.
  - ChauffeurNet is mid-to-mid (DNN used for behavior prediction and planning to generate trajectory).
- DNN Architecture
- Compare to PilotNet
  - PilotNet uses CNN to map from current camera image to steering angle, no history info is used.
  - ChauffeurNet uses RNN, and exploits the history (past agent location and past prediction for a few seconds) to make better predictions and decisions.
- CIL: adds a condition variable to help with the decision process at intersections.