L8 Autonomous Driving with IL&RL

Zonghua Gu 2021

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

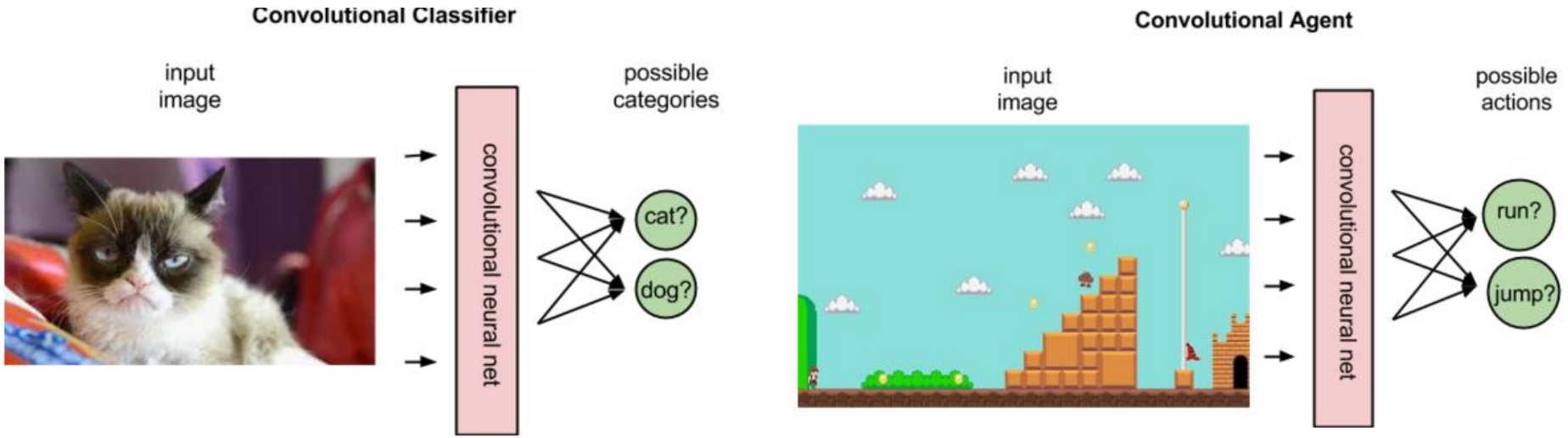
Notations Cont'd

- Rollout: sequentially execute policy $\pi_\theta(s_0)$ from initial state s_0 until timestep T.
  - Produce trajectory $\boldsymbol{\tau} = (s_0, a_0, s_1, a_1, \dots)$
- $P(\boldsymbol{\tau}|\pi)$: distribution of trajectories induced by a policy. Repeat until $t = T$:
  - 1. Sample s_0 from P_0 (distribution over initial states), initialize $t = 1$
  - 2. Sample action a_t from $\pi(s_{t-1})$
  - 3. Sample next state s_t from applying a_t to s_{t-1} (requires access to env model or simulator)
  - 4. Go to Step 2 with $t = t + 1$
- $P(s|\pi)$: distribution of states induced by a policy (percentage of time spent in each state):
  - $P(s|\pi) = \frac{1}{T}\sum_t P_t(s|\pi)$
    - $P_t(s|\pi)$ denotes distribution of states at t-th timestep

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).

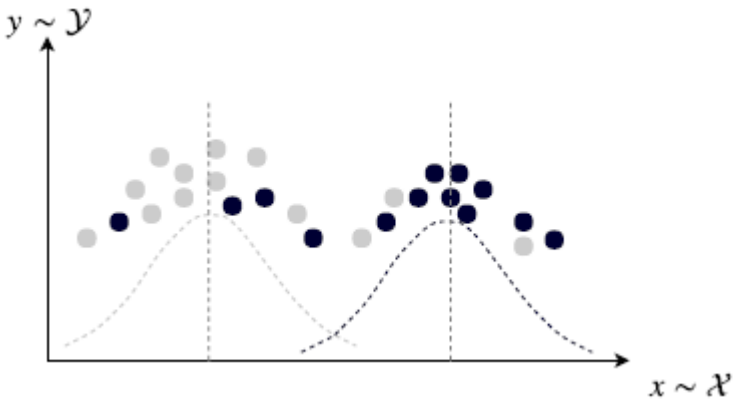
1. Collect demonstrations (τ^* trajectories) from expert
2. Treat the demonstrations as i.i.d. state-action pairs: $(s_0^*, a_0^*), (s_1^*, a_1^*), \ldots$
3. Learn π_θ policy using supervised learning by minimizing the loss function $L\left(a^*, \pi_\theta(s)\right)$



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.



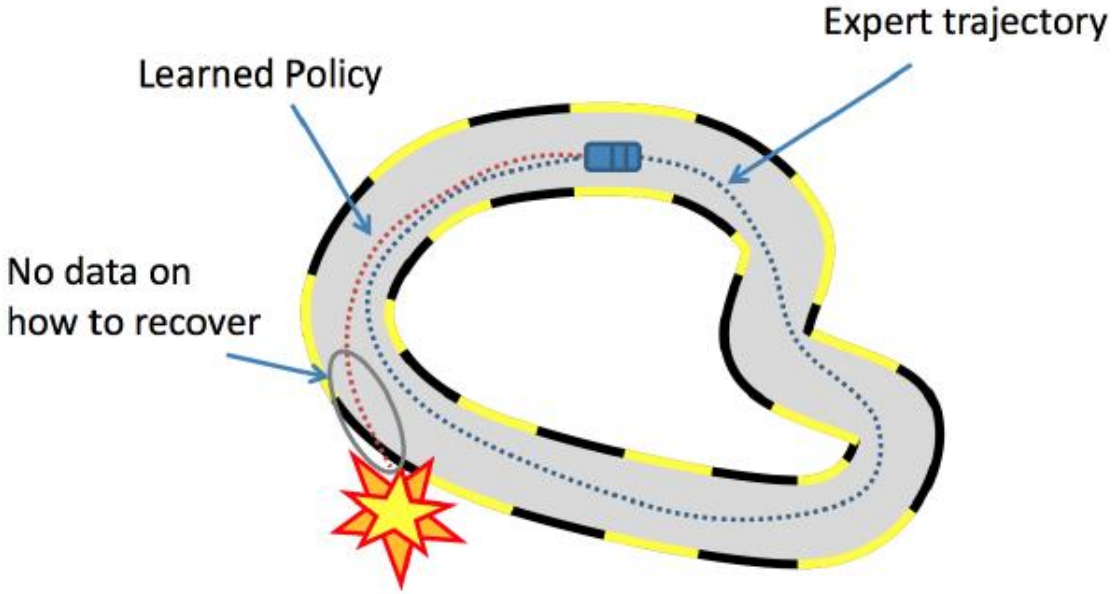
https://pathmind.com/wiki/deep-reinforcement-learning

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

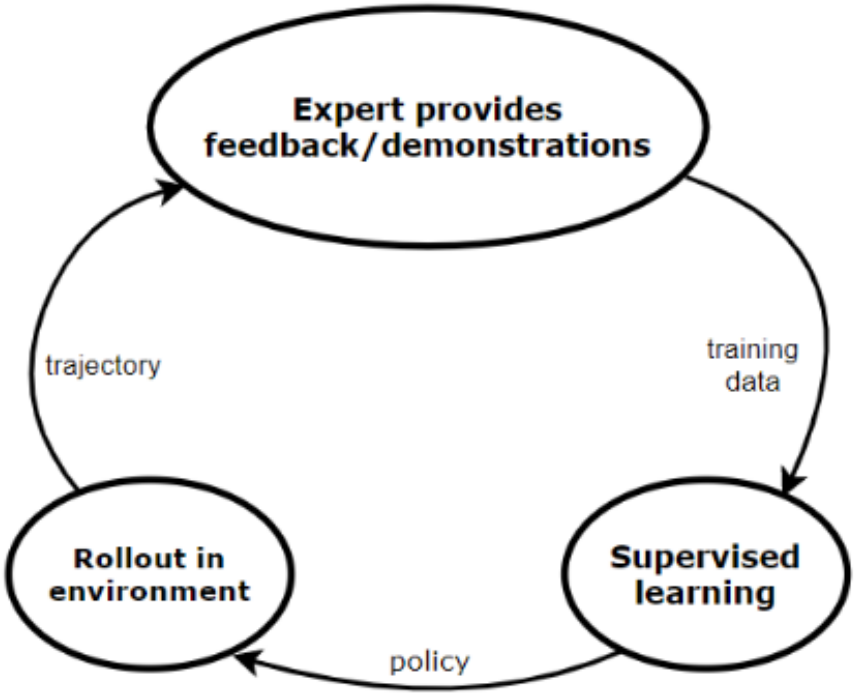
- Errors made in different states add up, therefore a mistake made by the agent can easily put it into a state that the expert has never visited and the agent has never trained on. In such states, the action is undefined and this can lead to catastrophic failures.
- e.g. the expert driver always keeps 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

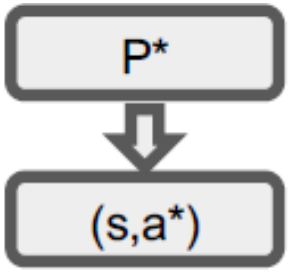
Direct Policy Learning (DPL)

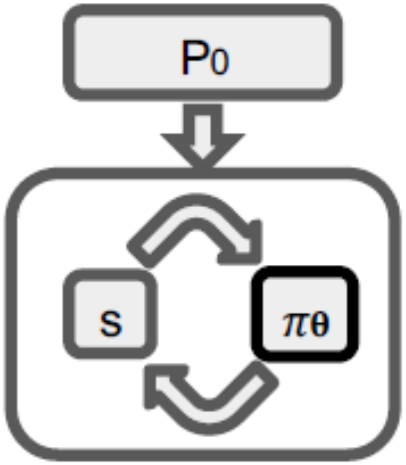
- DPL is an improved version of behavioral cloning. Assuming that we have access to an interactive expert at training time,
- First, we start with an initial predictor policy based on the initial expert demonstrations.
- Then, we execute a loop until we converge. In each iteration, we collect trajectories by rolling out the current policy (which we obtained in the previous iteration) and using these we estimate the state distribution.
- Then, for every state, we collect feedback from the expert (what would have he done in the same state).
- Finally, we train a new policy using this feedback.



BC vs. DPL

- BC: $\text{argmin}_\theta \, E_{(s,a^*)\sim P^*} \, L(a^*, \pi_\theta(s))$
  - Assuming perfect imitation so far ($(s, a^*) \sim P^*$), learn to continue imitating perfectly, where $P^* = P(s|\pi^*)$ (distribution of states visited by expert)
  - Minimize 1-step deviation error along expert trajectories
  - Distribution provided exogenously from expert demos
    - Agent is a passenger. Its own actions are never carried out. At every timestep, it computes the loss function that measures difference between its own action by its policy at current timestep with that of the expert.
- DPL:
  - $\text{argmin}_\theta \, E_{s\sim P(s|\pi_\theta)} \, L(\pi^*(s), \pi_\theta(s))$
  - Distribution depends on agent's rollout based on current policy π_θ
    - Agent is the driver that carries out its own action at every timestep, while computing the loss function that measures difference between its action/policy at current timestep with that of the expert.





Two Variants of DPL

- Data Aggregation trains the actual policy on all the previous training data.
  - e.g., Dagger: expert remains in the loop during the training of the controller: the controller is iteratively tested and samples from the obtained trajectories are re-labeled by the expert
- Policy Aggregation trains a policy on the training data received on the last iteration and then combines this policy with all the previous policies using geometric blending. In the next iteration, we use this newly obtained, blended policy during the roll-out

Initial predictor: π_0

For $m = 1$:

- Collect trajectories τ by rolling out π_{m-1}
- Estimate state distribution P_m using $s \in \tau$
- Collect interactive feedback $\{\pi^*(s) \mid s \in \tau\}$
- Data Aggregation (e.g. Dagger)
  - Train π_m on $P_1 \cup \ldots \cup P_m$
- Policy Aggregation (e.g. SEARN & SMILe)
  - Train π'_m on P_m
  - $\pi_m = \beta\pi'_m + (1-\beta)\pi_{m-1}$

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

Behavioral Cloning

$$\text{argmin}_\theta \, E_{(s,a^*)\sim P^*} L(a^*, \pi_\theta(s))$$

Works well when P^* close to P_θ

Inverse RL

Learn r such that:

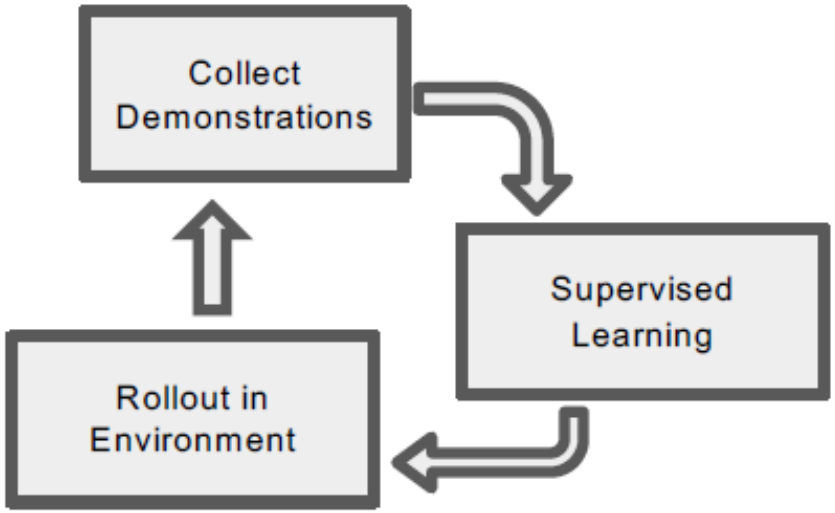
$$\pi^* = \text{argmax}_\theta \, E_{s\sim P(s|\theta)} r(s, \pi_\theta(s))$$

RL problem

Assumes learning r is statistically easier than directly learning π^*

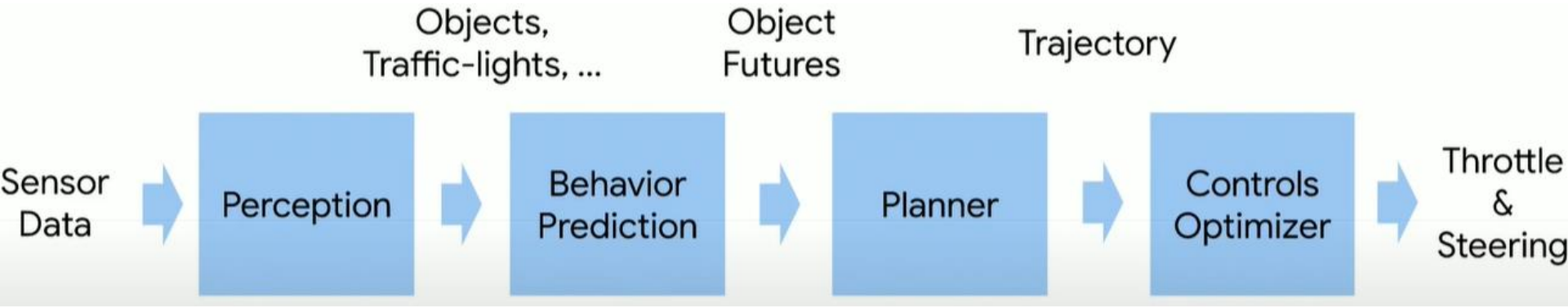
Direct Policy Learning

via Interactive Demonstrator



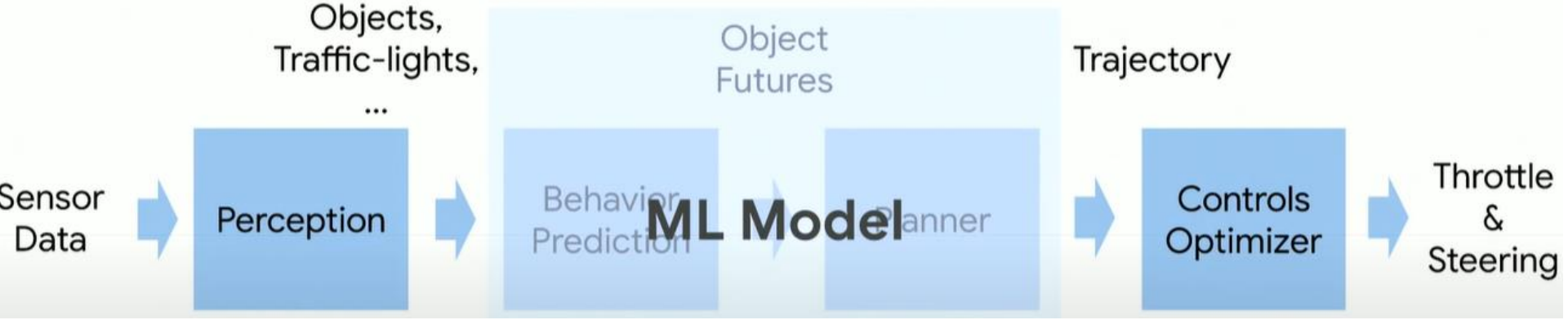
Requires Interactive Demonstrator (BC is 1-step special case)

	Learn Policy Directly	Learn Reward Function	Access to Env.	Interaction Demos	Pre-collected demos
BC	Y	N	N	N	Y
DPL	Y	N	Y	Y	Optional
IRL	N	Y	Y	N	Y



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.

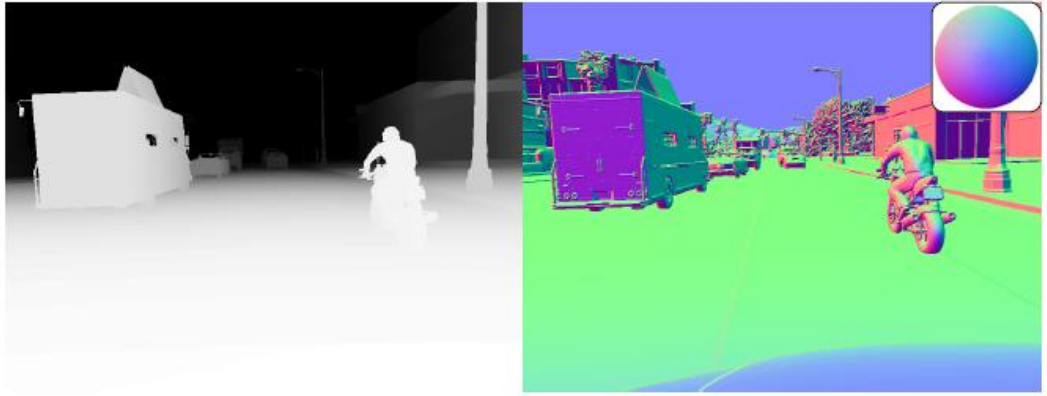
[Zhou 2019]

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

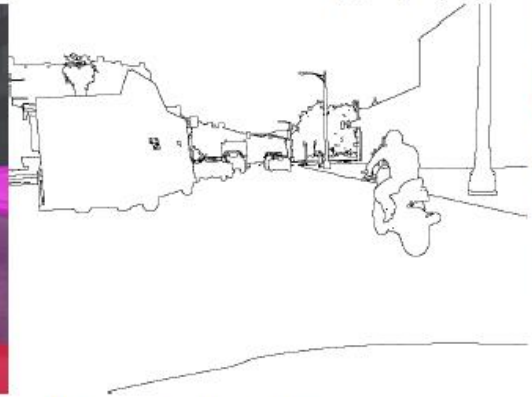
Computer Vision vs. End-to-End

- 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

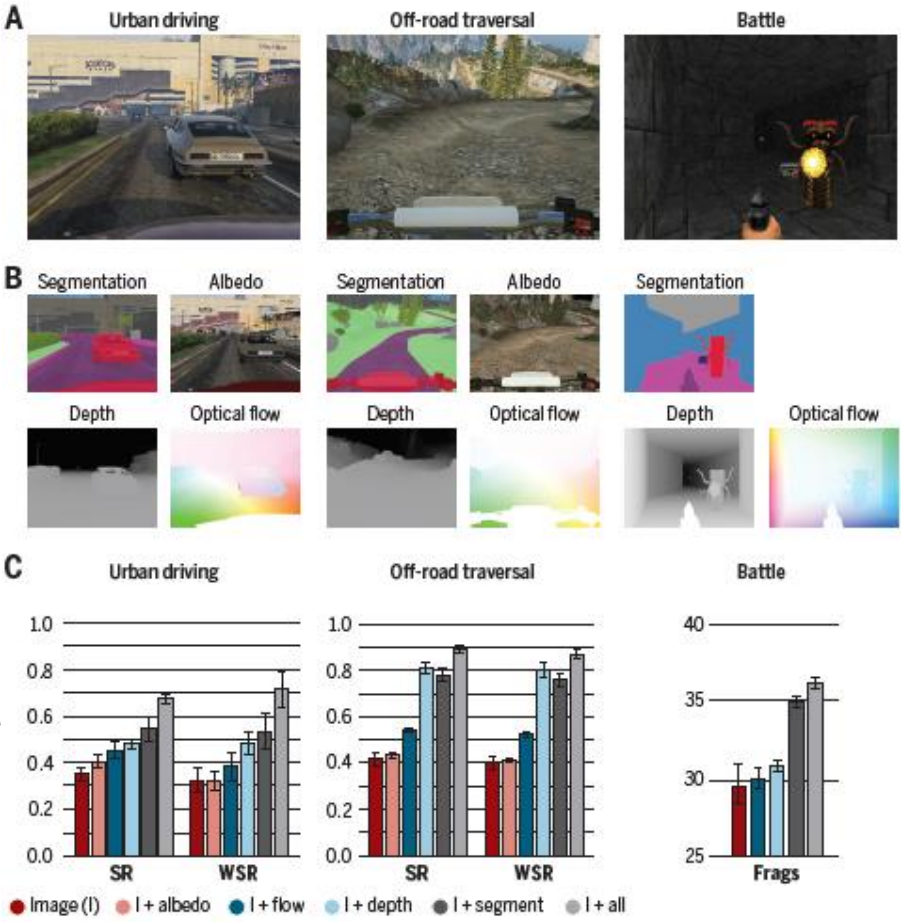


Fig. 1. Assessing the utility of intermediate representations for sensorimotor control. (**A**) Sensorimotor tasks. From left to right: urban driving, off-road trail traversal, and battle. (**B**) Intermediate representations. Clockwise from top left: semantic segmentation, intrinsic surface color (albedo), 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)

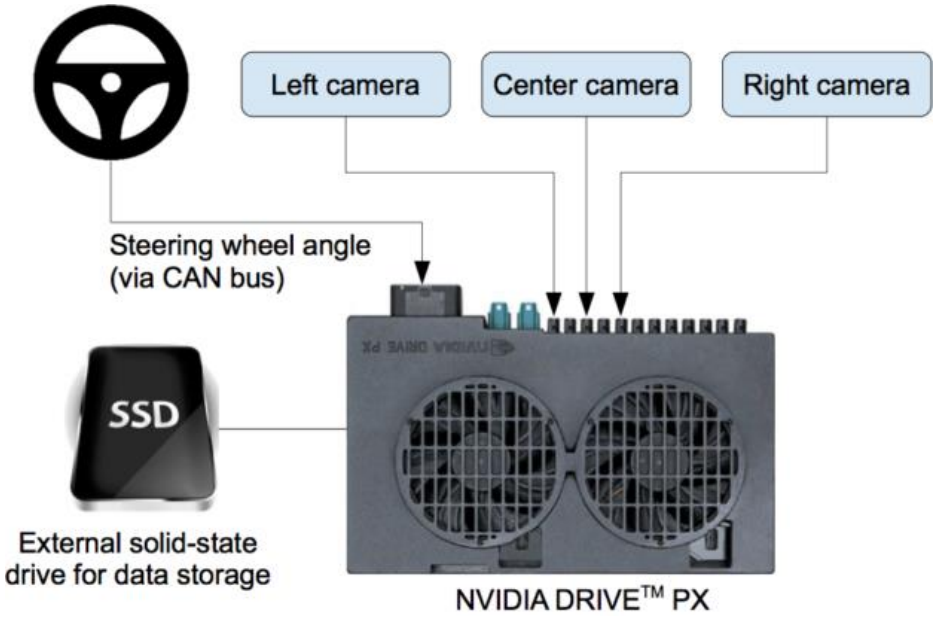
Training Method

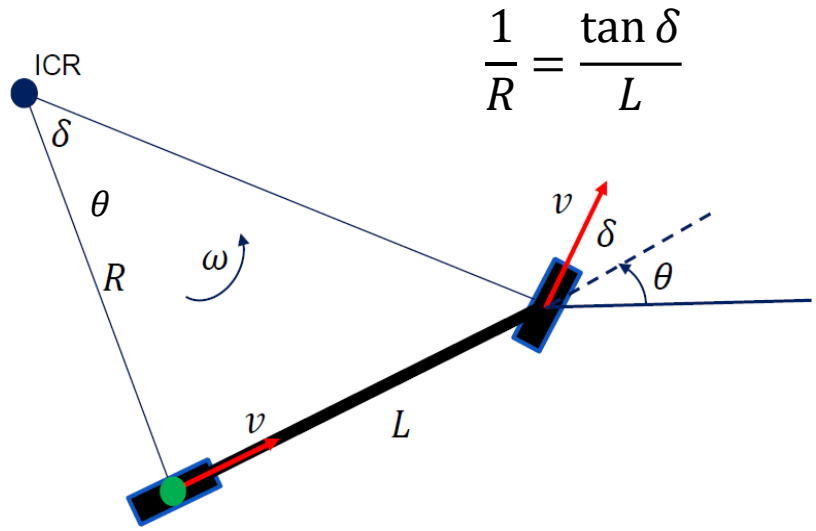
- End-to-End model trained with Imitation Learning (BC)
  - Human drives vehicle
  - Record sensor data and human actuator commands as training pairs
  - Train a DNN to map sensor data to actuator commands, mimicking a human. (PilotNet controls steering only. I think acceleration/braking are controlled 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

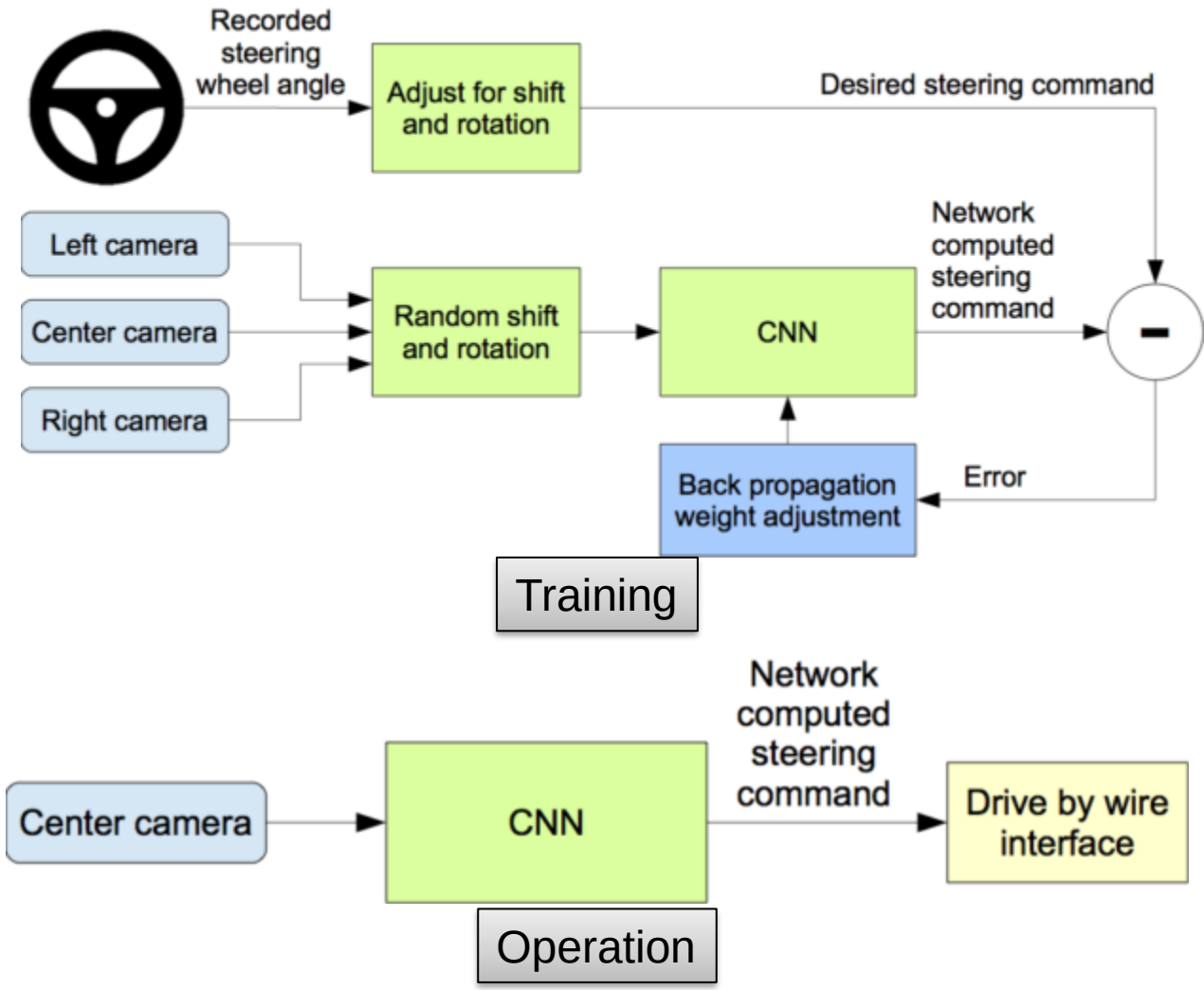
- Three cameras are mounted behind the windshield of the data-acquisition car, and timestamped video from the cameras is captured simultaneously with the steering angle applied by the human driver. The steering command is obtained by tapping into the vehicle's Controller Area Network (CAN) bus.
- In 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 ($\frac{1}{R}$).





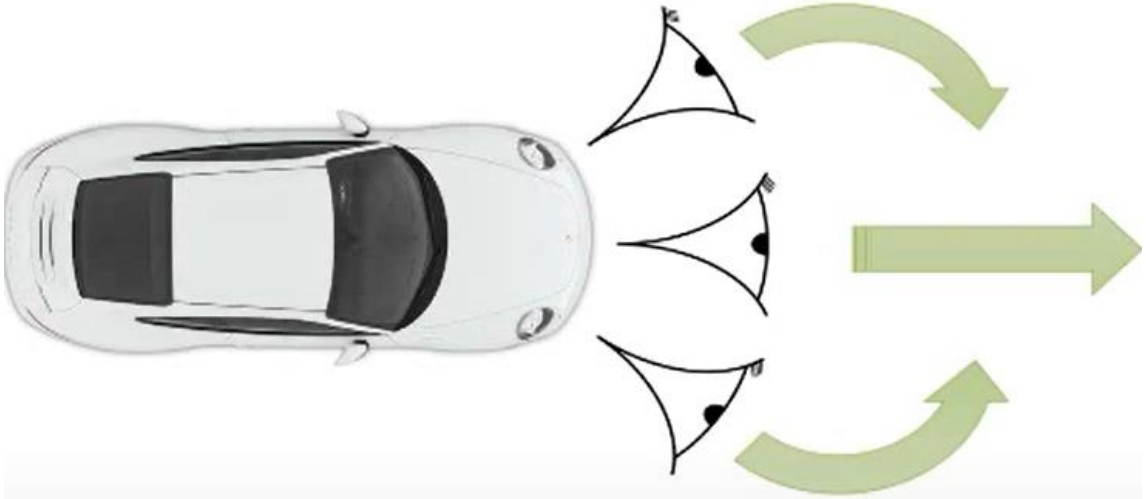
Training Data Augmentation

- 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.

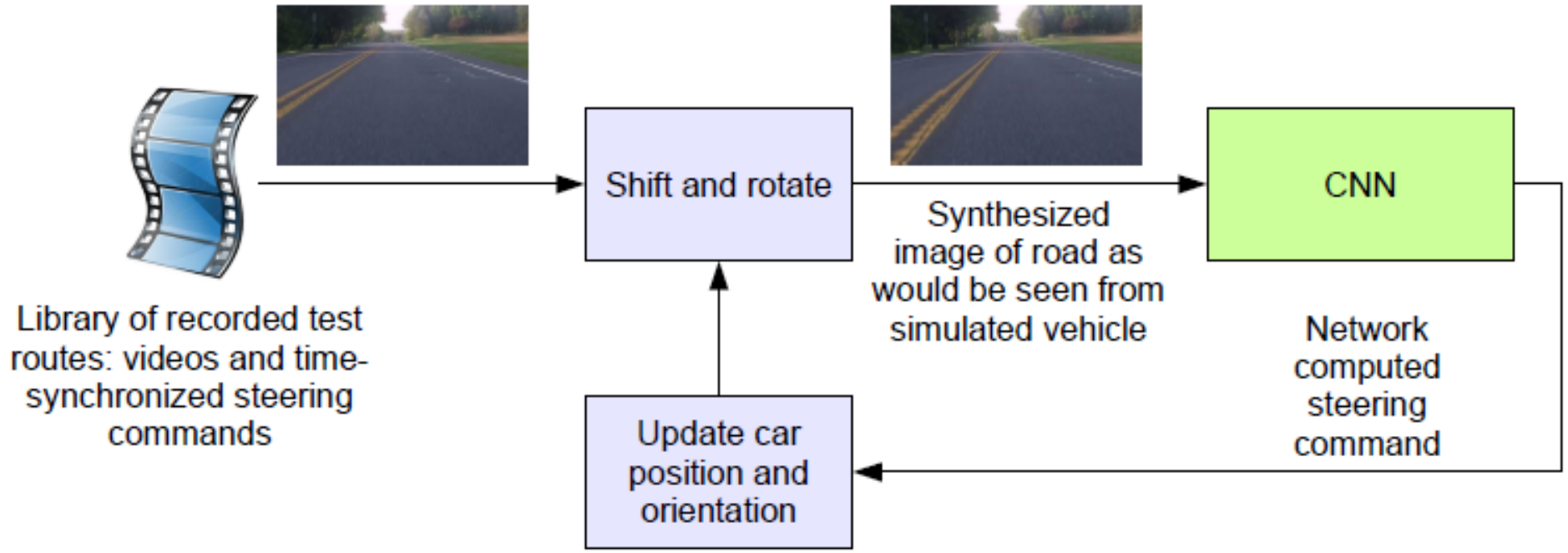


Figure 5: Block-diagram of the drive simulator.

PilotNet

- ~250K distinct weights
- ~27M connections

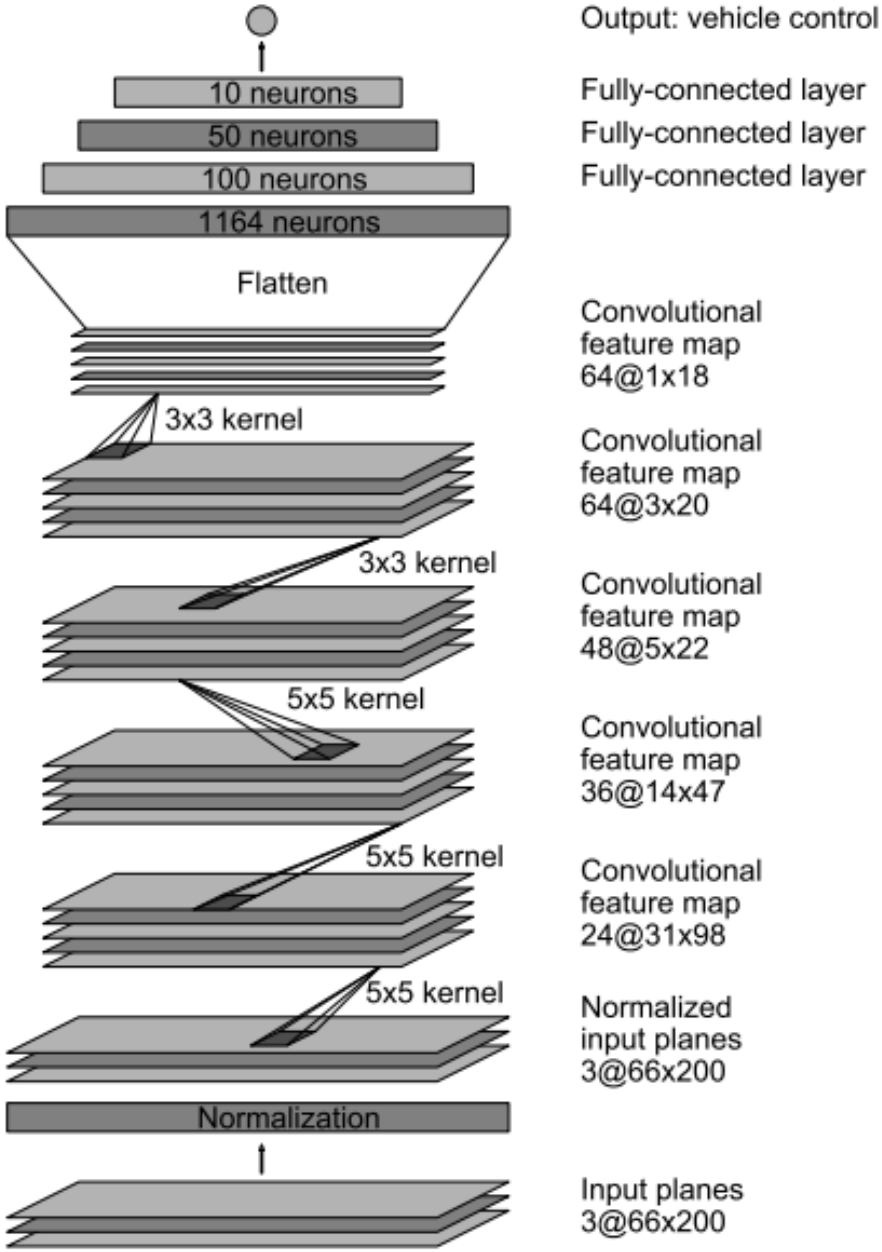


Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters. PilotNet

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

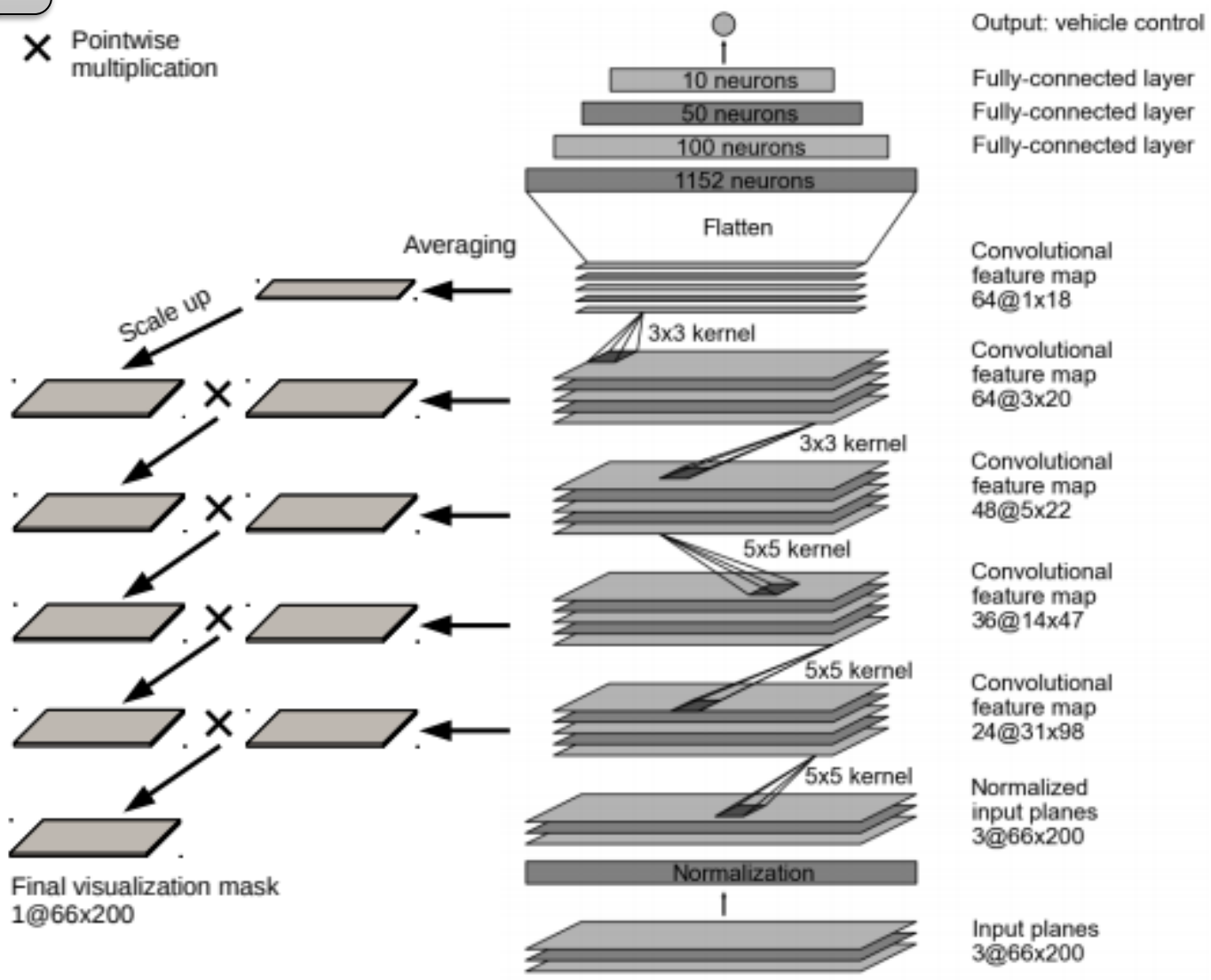


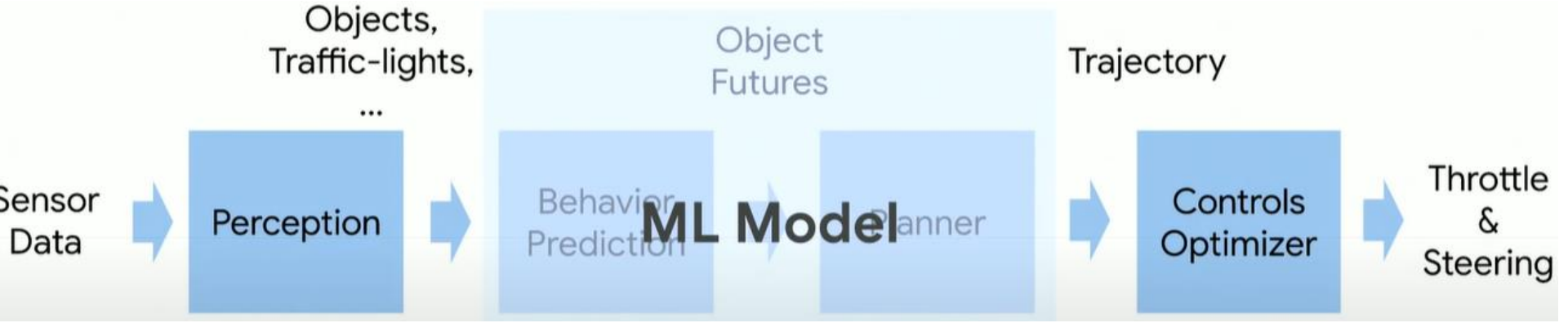
Figure 2: Block diagram of the visualization method that identifies the salient objects.

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.

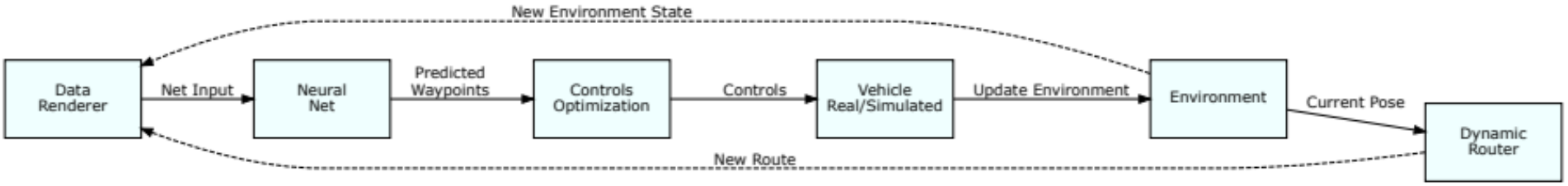
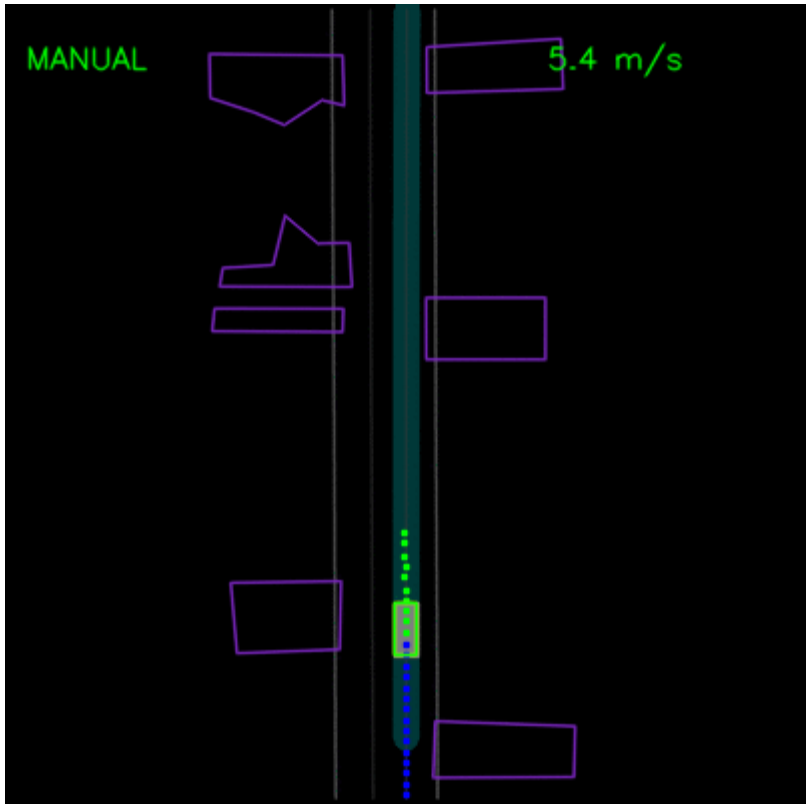


Figure 4: Software architecture for the end-to-end driving pipeline.

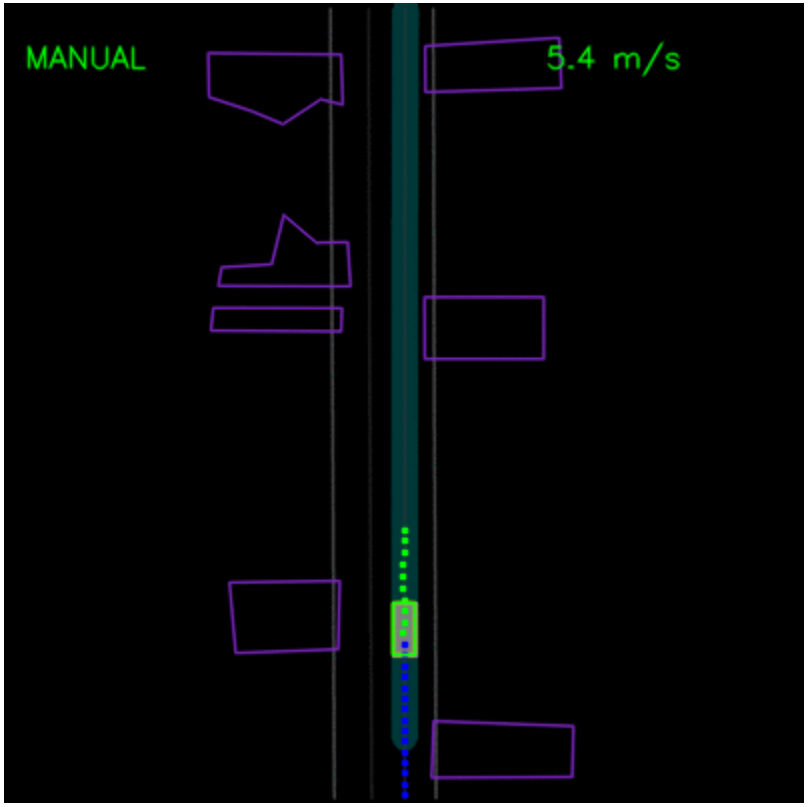
BC Experiments

- 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.



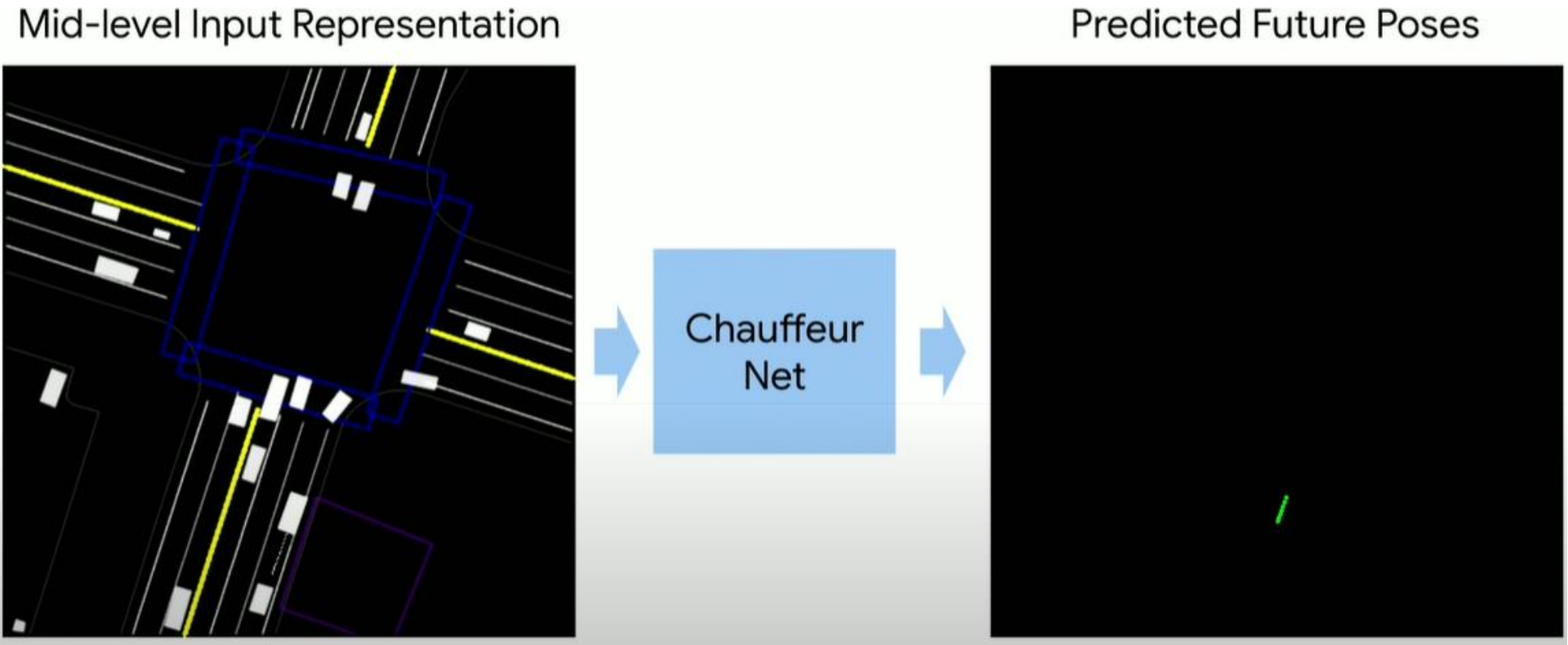
ChauffeurNet Experiments

- *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).

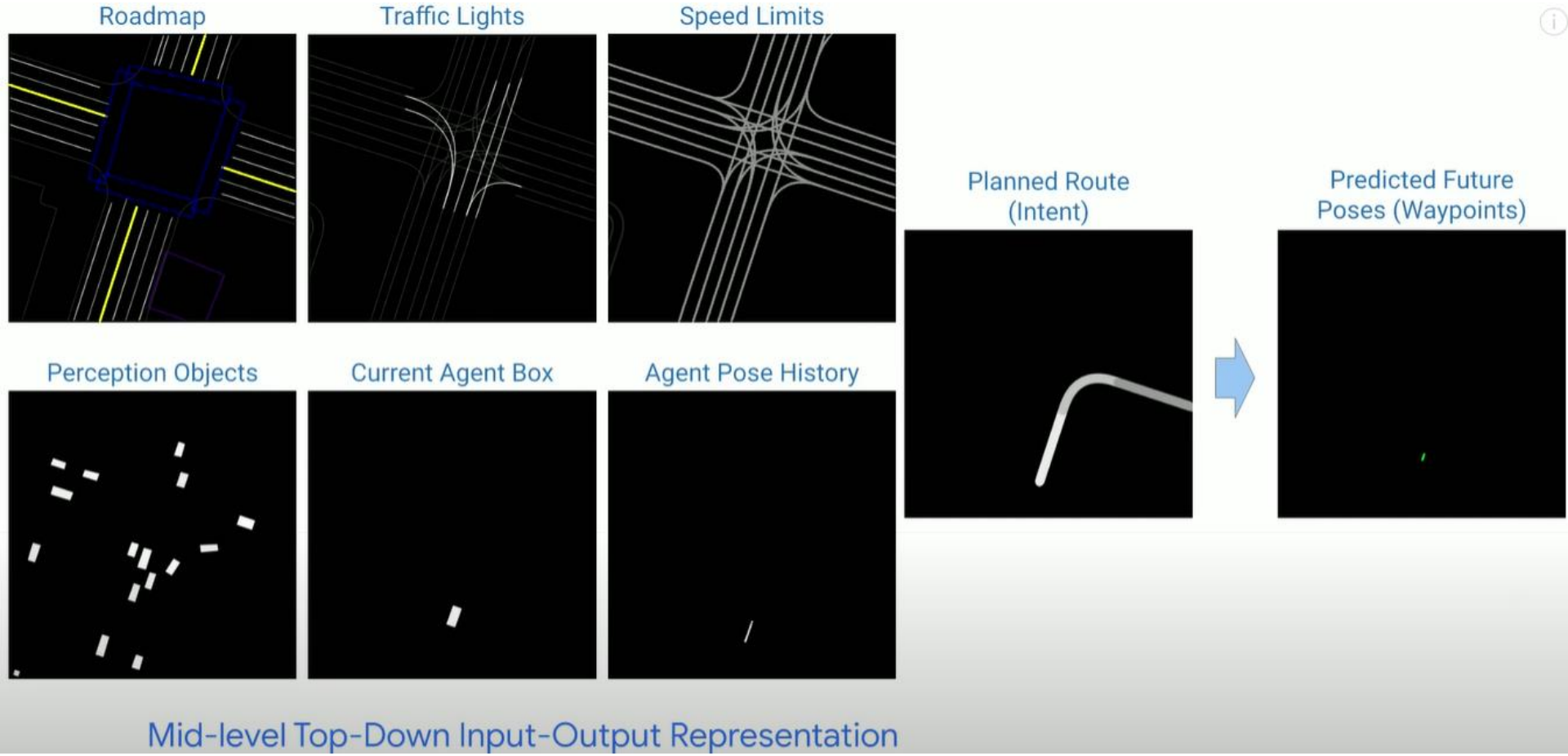


ChauffeurNet Input/Output

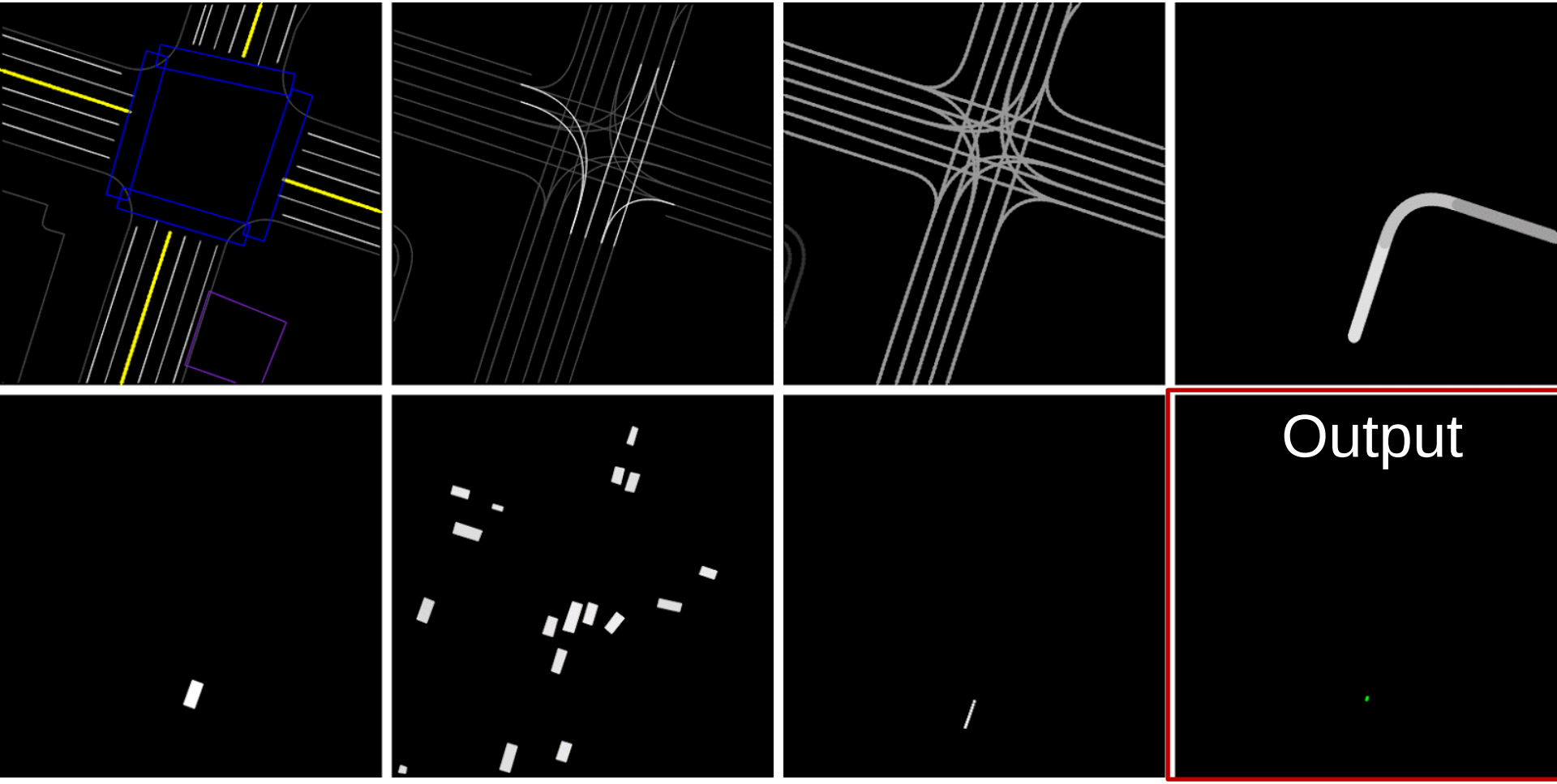
- We use a perception system that processes raw sensor information and produces our input: a top-down representation of the environment and intended route, where objects such as vehicles are drawn as oriented 2D boxes along with a rendering of the road information and traffic light states. We present 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
- Compare to PilotNet
  - PilotNet uses CNN to map from current camera image to steering angle, no history info is use.
  - ChauffeurNet uses RNN, and exploits the history (past agent location and past prediction for a few seconds) to make better predictions and decisions.

Not covered in exam

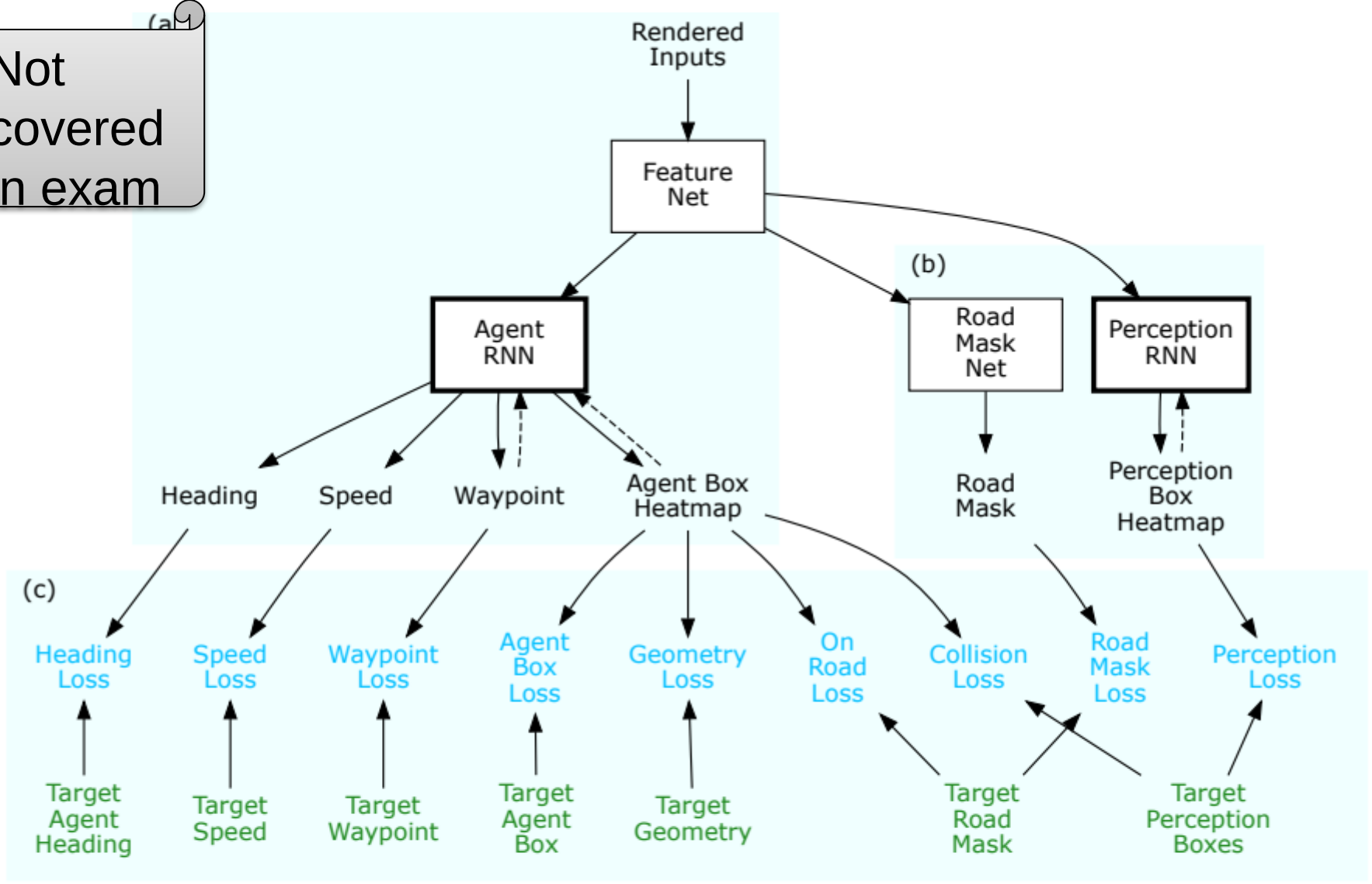


Figure 2: Training the driving model. (a) The core *ChauffeurNet* model with a FeatureNet and an AgentRNN, (b) Co-trained road mask prediction net and PerceptionRNN, and (c) Training losses are shown in blue, and the green labels depict the ground-truth data. The dashed arrows represent the recurrent feedback of predictions from one iteration to the next.

Not covered in exam

Fig. 3 illustrates the ChauffeurNet model in more detail. The rendered inputs shown in Fig. 1 are fed to a large-receptive field convolutional *FeatureNet* with skip connections, which outputs features F that capture the environmental context and the intent. These features are fed to the *AgentRNN* which predicts the next point $\mathbf{p}_k$ on the driving trajectory, and the agent bounding box heatmap B_k, conditioned on the features F from the FeatureNet, the iteration number $k \in \{1, \ldots, N\}$, the memory M_{k-1} of past predictions from the *AgentRNN*, and the agent bounding box heatmap B_{k-1} predicted in the previous iteration.

$$\mathbf{p}_k, B_k = \text{AgentRNN}(k, F, M_{k-1}, B_{k-1}) \tag{2}$$

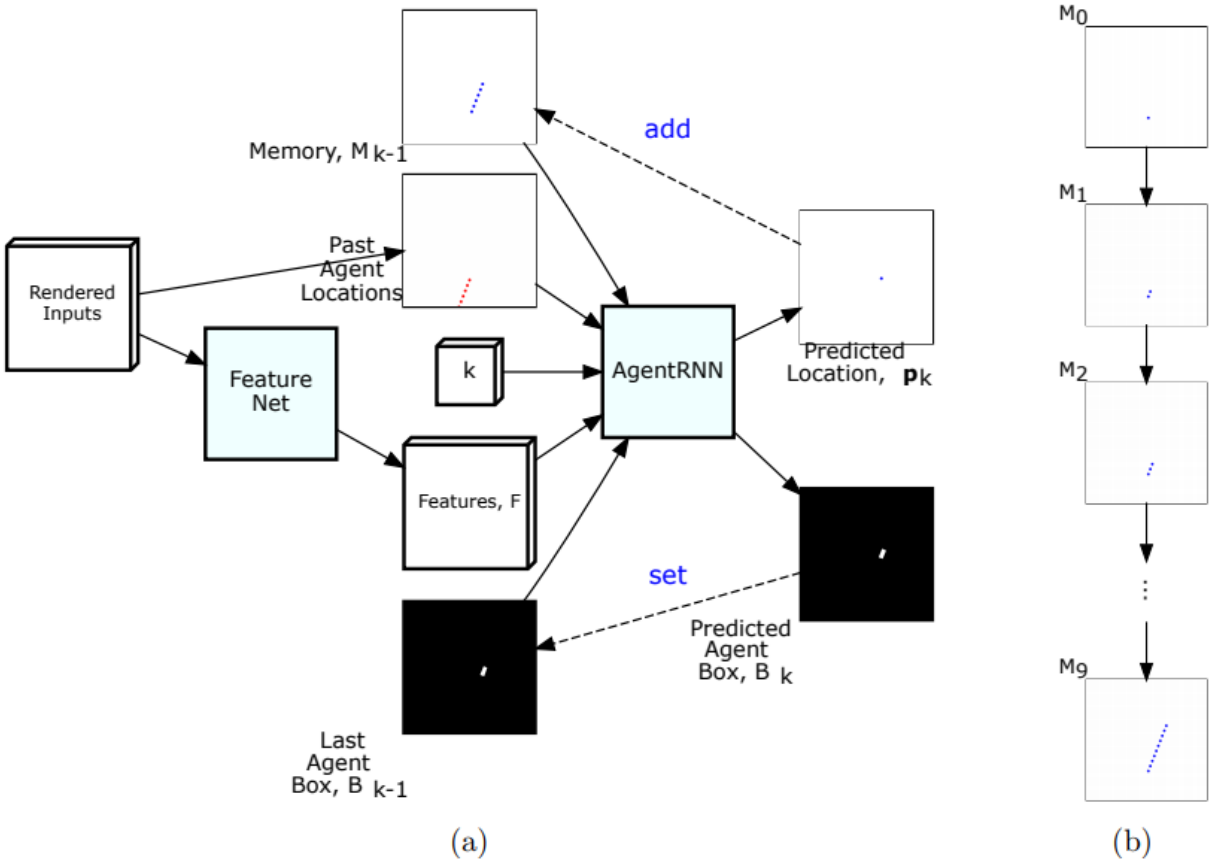


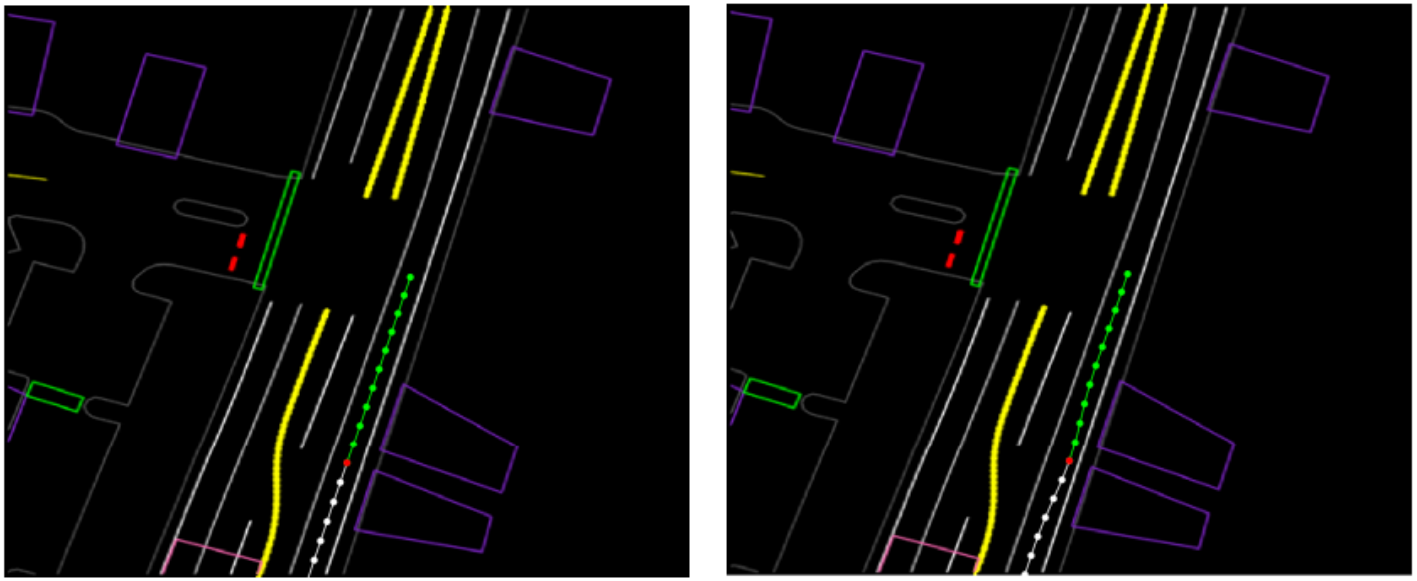
Figure 3: (a) Schematic of ChauffeurNet. (b) Memory updates over multiple iterations.

Beyond Pure Imitation

- Synthesizing Perturbations
- Beyond the Imitation Loss
  - Collision Loss
  - On Road Loss
  - Geometry Loss
  - Auxiliary Losses
- Imitation Dropout

Trajectory Perturbation

- Running the model as a part of a closed-loop system over time can cause the input data to deviate from the training distribution. To prevent this, we train the model by adding some examples with realistic perturbations to the agent trajectories.



(a) Original (b) Perturbed

Figure 5: Trajectory Perturbation. (a) An original logged training example where the agent is driving along the center of the lane. (b) The perturbed example created by perturbing the current agent location (red point) in the original example away from the lane center and then fitting a new smooth trajectory that brings the agent back to the original target location along the lane center.

Collision Loss

- Since our training data does not have any real collisions, the idea of avoiding collisions is implicit and will not generalize well. To alleviate this issue, we add a specialized loss that directly measures the overlap of the predicted agent box B_k with the ground-truth boxes of all the scene objects at each timestep
- (Details of On Road Loss, Geometry Loss, Auxiliary Losses omitted.)

$$\mathcal{L}_{collision} = \frac{1}{WH} \sum_x \sum_y B_k(x,y) \; . \; Obj_k^{gt}(x,y) \tag{8}$$

where B_k is the likelihood map for the output agent box prediction, and Obj_k^{gt} is a binary mask with ones at all pixels occupied by other dynamic objects (other vehicles, pedestrians,

Not covered in exam

Imitation Dropout

3.3 Imitation Dropout

Overall, our losses may be grouped into two sub-groups, the imitation losses:

$$\mathcal{L}_{imit} = \{\mathcal{L}_p, \mathcal{L}_B, \mathcal{L}_\theta, \mathcal{L}_{p-subpixel}, \mathcal{L}_{speed}\} \tag{13}$$

and the environment losses:

$$\mathcal{L}_{env} = \{\mathcal{L}_{collision}, \mathcal{L}_{onroad}, \mathcal{L}_{geom}, \mathcal{L}_{objects}, \mathcal{L}_{road}\} \tag{14}$$

The imitation losses cause the model to imitate the expert's demonstrations, while the environment losses discourage undesirable behavior such as collisions. To further increase the effectiveness of the environment losses, we experimented with randomly dropping out the imitation losses for a random subset of training examples. We refer to this as "imitation dropout". In the experiments, we show that imitation dropout yields a better driving model than simply under-weighting the imitation losses. During imitation dropout, the weight on the imitation losses w_{imit} is randomly chosen to be either 0 or 1 with a certain probability for each training example. The overall loss is given by:

$$\mathcal{L} = w_{imit} \sum_{\ell \in \mathcal{L}_{imit}} \ell + w_{env} \sum_{\ell \in \mathcal{L}_{env}} \ell \tag{15}$$

Model	Description	w_{imit}	w_{env}
$\mathcal{M}_0$	Imitation with Past Dropout	1.0	0.0
$\mathcal{M}_1$	$\mathcal{M}_0$ + Traj Perturbation	1.0	0.0
$\mathcal{M}_2$	$\mathcal{M}_1$ + Environment Losses	1.0	1.0
$\mathcal{M}_3$	$\mathcal{M}_2$ with less imitation	0.5	1.0
$\mathcal{M}_4$	$\mathcal{M}_2$ with Imitation Dropout	*Dropout probability = 0.5 (see Section 5.3).*	

Table 3: Model configuration for the model ablation tests.

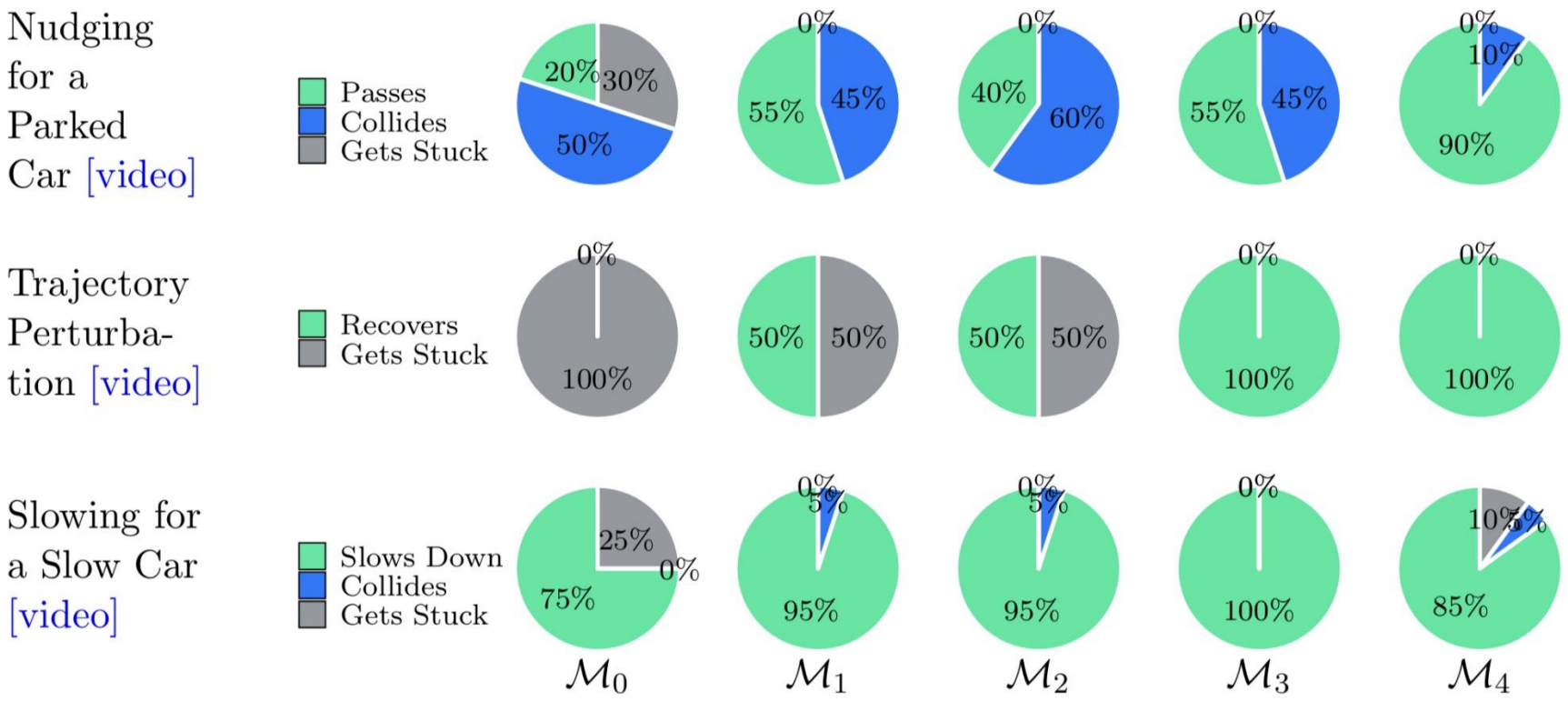
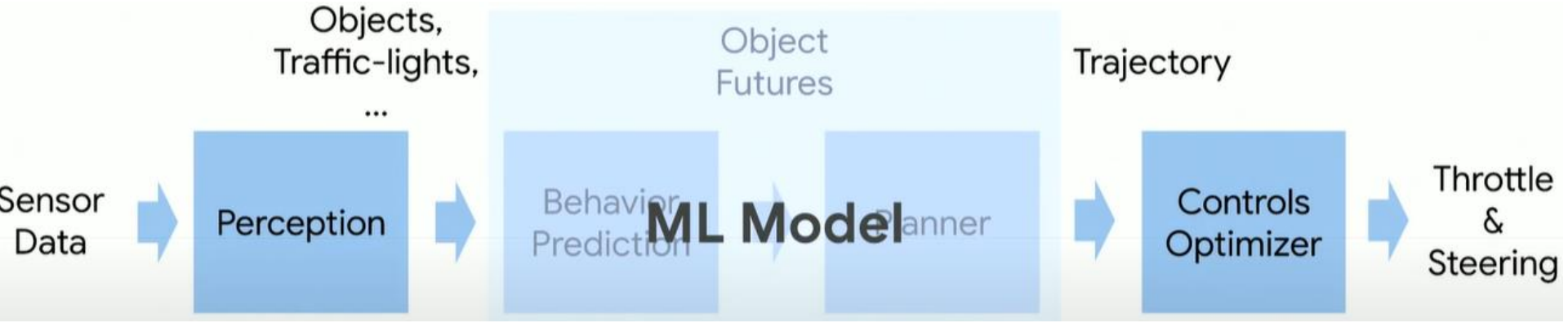


Figure 7: Model ablation test results on three scenario types.

IL with Safety [Chen 2019]

- Chen J, Yuan B, Tomizuka M. Deep imitation learning for autonomous driving in generic urban scenarios with enhanced safety[J]. arXiv preprint arXiv:1903.00640, 2019.

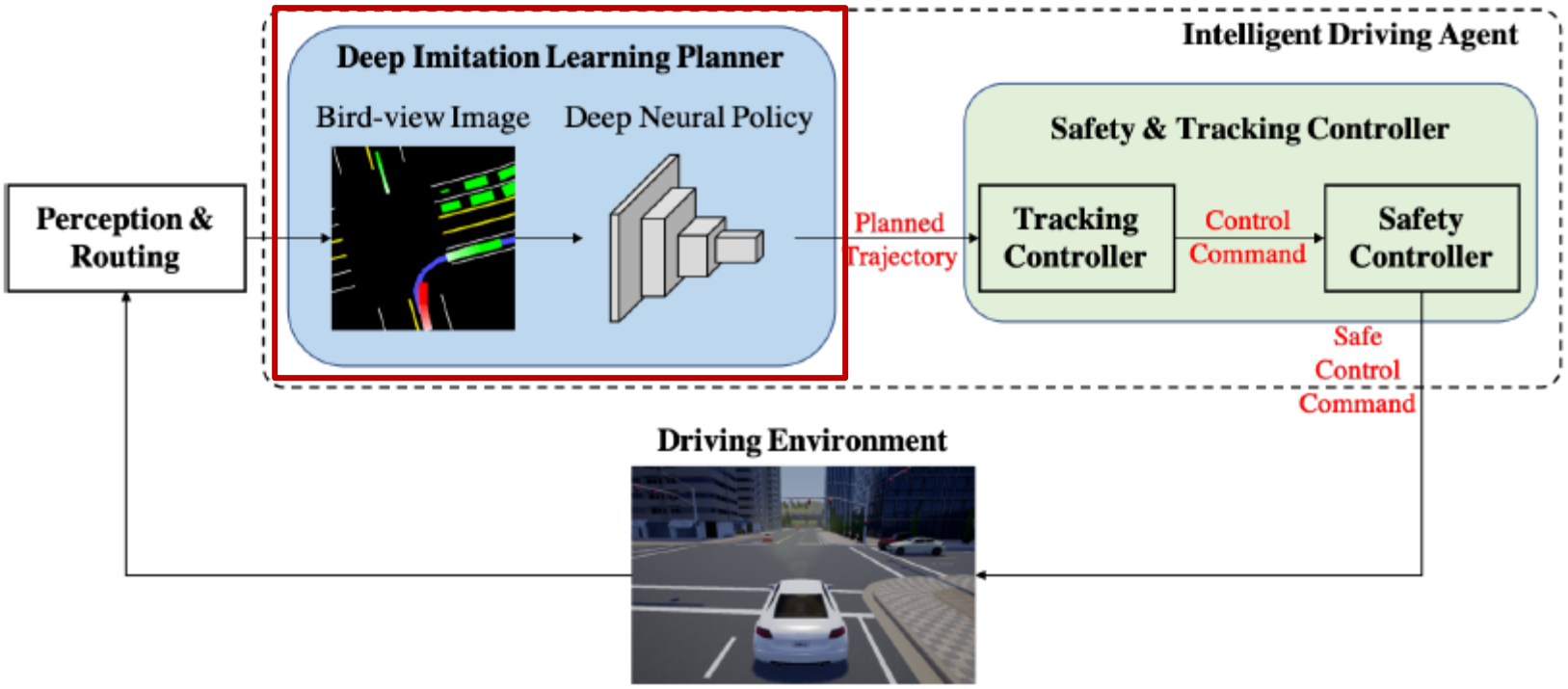


Abstract

- The decision and planning system for AD in urban environments is hard to design. Most current methods are to manually design the driving policy, which can be sub-optimal and expensive to develop and maintain at scale. Instead, with imitation learning we only need to collect data and then the computer will learn and improve the driving policy automatically. However, existing imitation learning methods for autonomous driving are hardly performing well for complex urban scenarios. Moreover, the safety is not guaranteed when we use a DNN policy. In this paper, we proposed a framework to learn the driving policy in urban scenarios efficiently given offline collected driving data, with a safety controller incorporated to guarantee safety at test time. The experiments show that our method can achieve high performance in realistic three-dimensional simulations of urban driving scenarios, with only hours of data collection and training on a single consumer GPU.

Framework Overview

- The agent takes information from the perception and routing modules, generates a bird-view image and outputs the planned trajectory using a DNN policy network. The safety & tracking controller then calculates the safe control command to be applied to the ego vehicle in the driving environment



Deep Imitation Learning

- DNN output is future trajectory in a preview horizon H: $[x_{t+1}, y_{t+1}, \dots, x_{t+H}, y_{t+}$, which is tracked by a low-level controller (PID, MPC).

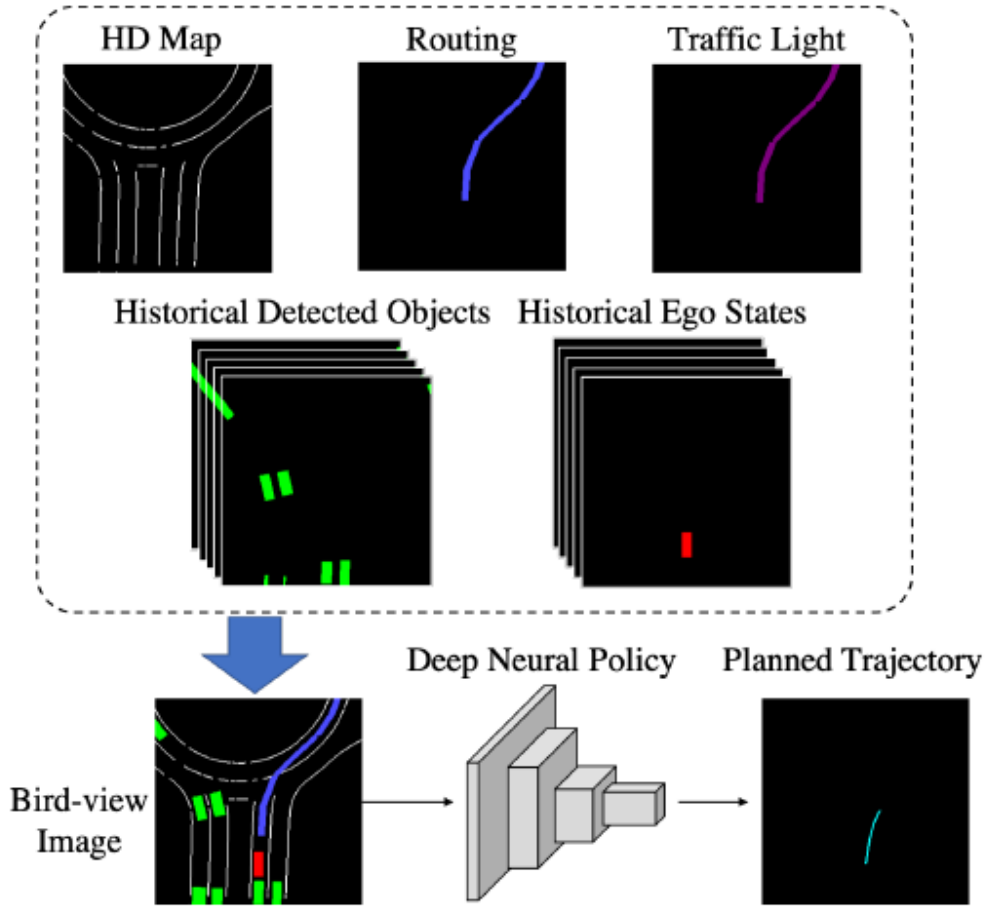


Fig. 2: *Observation-action Representation of our deep imitation learning planner. The bird-view observation combines information of HD map, routing, traffic light, historical detected objects and historical ego states. The output action is a planned trajectory represented by a vector.*

DNN Architecture & Loss Function

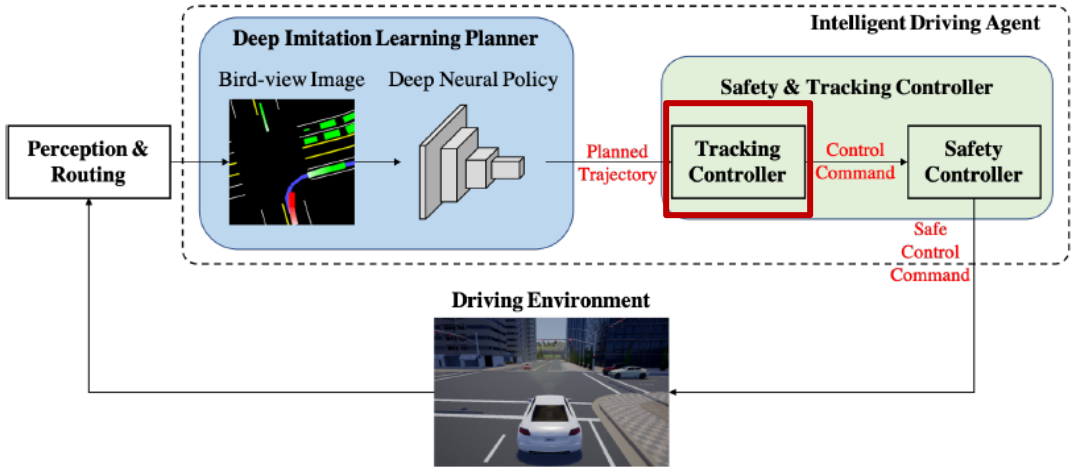
- Based on CNN (VGGNet16). Output layer has H units corresponding to H predicted trajectory points $(\hat{x}_{t+i}, \hat{y}_{t+i}), i \in [1, H]$.
- Loss function to be minimized:
  - $L_t = \frac{1}{H}\sum_{i=1}^{H} d_{t+i}^2$
  - where displacement error d_{t+i} between the expert's motion trajectory point position (x_{t+i}, y_{t+i}) and the predicted point position $(\hat{x}_{t+i}, \hat{y}_{t+i})$: $d_{t+i} = \left((x_{t+i} - \hat{x}_{t+i})^2 + (y_{t+i} - \hat{y}_{t+i})^2\right)^{\frac{1}{2}}$

Data Collection and Augmentation

- Based on CARLA simulator.
- At data collection phase, we use a model-based controller to act as the expert. The controller is same as other agents, which performs normal urban driving behaviors and make random turns at intersections. When ego vehicle is running, we record the rendered bird-view images and the corresponding ego vehicle states (global positions and yaw angle) every 0.1 second.
- We introduce control noise to the expert controller during the data collection phase, and let the expert recover from the perturbation. The control noise is added periodically every 8 seconds, and will last for 1 second. The vehicle's pose might be pushed away from the way points. The expert then provides demonstrations of recovering from perturbations. The states during the noise phase are removed in order not to contaminate the dataset

Trajectory Tracking Controller

- Given the planned future trajectory $[\hat{x}_{t+1}, \hat{y}_{t+1}, \dots, \hat{x}_{t+H}, \hat{y}_{t+H}]$, the tracking controller outputs the desired acceleration a_t and steering angle δ_t to drive the vehicle to follow the trajectory. A way point $(\hat{x}_{t+m}, \hat{y}_{t+m})$ is selected with $m \in [1, H-1]$ (we pick $m = 5$ here).
  - At any time instant t, aim for the 5th point $(\hat{x}_{t+5}, \hat{y}_{t+5})$ on the planned trajectory
- Decomposed into longitudinal controller and lateral controller, both PID controllers.



Not covered in exam

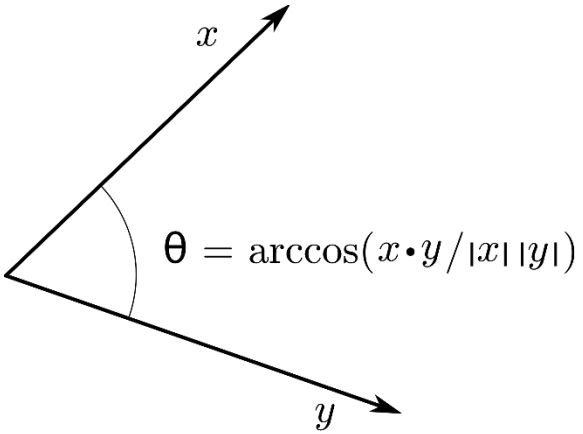
Longitudinal Controller

- Target speed is set to be:
  - $v_d = \frac{1}{dt} \|(\hat{x}_{t+m+1}, \hat{y}_{t+m+1}) - (\hat{x}_{t+m}, \hat{y}_{t+m})\|_2$
  - Where dt is the time interval between two subsequent time steps.
- The desired acceleration is obtained using PID control to minimize speed tracking error
  - $e_v(t) = v_d - v(t)$
  - where $v(t)$ is current speed of ego vehicle.

Not covered in exam

Lateral Controller

- The normal vector from the ego vehicle position to the target way point is
  - $\boldsymbol{n}_{target} = \frac{(\hat{x}_{t+m}, \hat{y}_{t+m})}{\|(\hat{x}_{t+m}, \hat{y}_{t+m})\|_2}$
- The normal vector of the ego vehicle heading is
  - $\boldsymbol{n}_{ego} = (\cos\theta_t, \sin\theta_t)$
  - where θ_t is the heading angle of the ego vehicle
- Then the desired steering angle is obtained using PID control to minimize the heading error:
  - $e_{yaw}(t) = \cos^{-1}(\boldsymbol{n}_{ego}(t) \cdot \boldsymbol{n}_{ego}(t))$
- Recall vector dot product: $x \cdot y = \|x\|\|y\| \cos\theta$

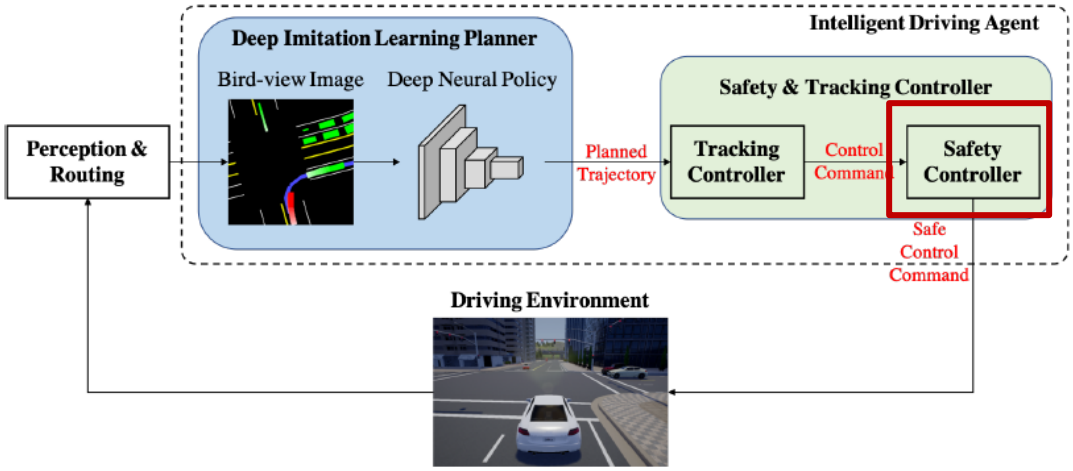


https://en.wikipedia.org/wiki/Dot_product

Not covered in exam

Safety Enhancement Controller

- The acceleration and steering command $\begin{bmatrix} a_t \\ \delta_t \end{bmatrix}$ calculated by the trajectory tracking controller does not guarantee safety, e.g, no collisions to other agents.
- Safety controller for collision avoidance will modify $\begin{bmatrix} a_t \\ \delta_t \end{bmatrix}$ to enhance safety, if their original values are not safe.
- The safe set algorithm: for each time step t, calculate a control safe set $U_S(t)$, s.t. if control command $u(t) = \begin{bmatrix} a_t \\ \delta_t \end{bmatrix} \in U_S(t)$, the ego vehicle would stay safe.



Not covered in exam

Safety Index

- Safety index $\phi(x)$: function of vehicle state x, which includes states (position, velocity, heading) of both ego vehicle (x_0) and a surrounding object (x_j). The system is considered safe if $\phi(x) \leq 0 \wedge d \geq 0$, and unsafe otherwise.
- $\phi(x) = D - d^2(x) - \alpha \dot{d}(x)$
- where $d(x) = \sqrt{[p_0 - p_j]^T Q [p_0 - p_j]}$ is shaped distance between ego vehicle and surrounding vehicle. p_0 indicates position of ego vehicle, and p_j indicates position of the surrounding object. Q is a 2×2 matrix s.t. $[p_0 - p_j]^T Q [p_0 - p_j] = 1$ encodes an ellipse around the surrounding vehicle with long axis equal to 1 and short axis equal to $\frac{1}{\beta}$, where β is the aspect ratio of the ellipse. α is a tunable parameter.

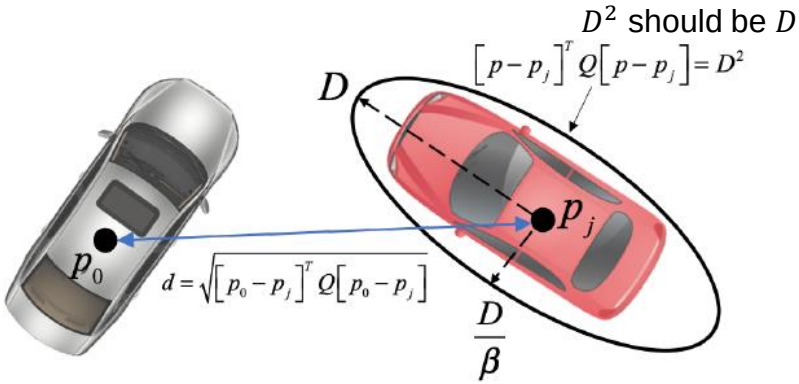
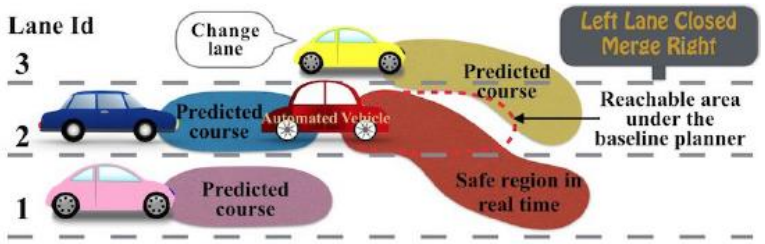
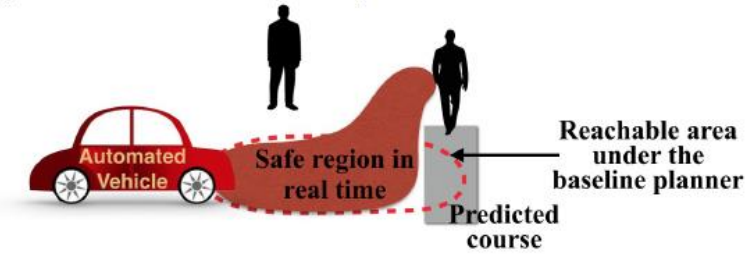


Fig. 3: *Illustration of the safety index. Gray is the ego vehicle, red is a surrounding vehicle. The safety constraint is similar to the ellipse around the red vehicle, while also considering the relative speed of the two vehicles.*



(a) Illustration of the freeway driving scenario.

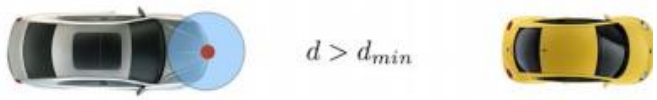


(b) Illustration of the scenario in an unstructured environment.
Figure 2. The function of the ROAD system.

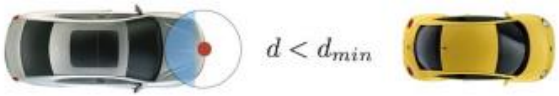
Not covered in exam

Safety Index Illustration

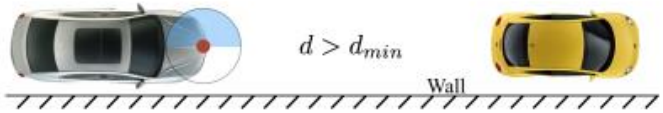
- $\phi(x) = D - d^2(x) - \alpha\dot{d}(x)$
- $\phi(x) \leq 0 \Rightarrow d^2(x) + \alpha\dot{d}(x) \geq D$
- The larger the distance $d(x)$, the larger relative speed $\dot{d}(x)$ (positive speed means moving away from each other), the safer it is.



(b) U_S when the distance is large enough ($\phi < 0$)

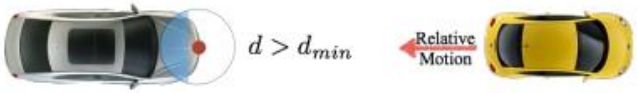


(c) U_S when the distance is small ($\phi > 0$)

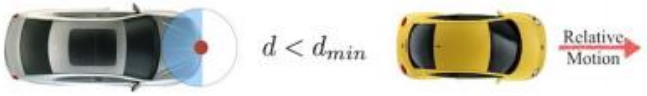


(d) U_S with boundary constraint ($\phi < 0,\ \phi_{wall} = 0$)

Figure 7. The safety constraint U_S with respect to a relatively static front vehicle.



(a) U_S with decreasing relative distance ($\phi > 0$)



(b) U_S with increasing relative distance ($\phi > 0$)

Figure 8: The safety constraint U_S with respect to a relatively moving front vehicle.

Not covered in exam

State Safe Set and Control Safe Set

- State safe set X_S: level set of the safety index
  - $X_S = \{x: \phi(x) \leq 0\} = \{x: d^2(x) + \alpha \dot{d}(x) \geq D\}$
  - It injects an ellipse constraint as shown in Fig. 3, also considering the relative speed between the ego and surrounding object.
- Control safe set
  - $U_S(t) = \{u(t): \dot{\phi} \leq -\eta \text{ if } \phi \geq 0\}$ where $\eta > 0$ is some margin. It is easy to prove that if $x(0) \in X_S \wedge u(t) \in U_S$ for $t \geq 0$, then $x(t) \in X_S$
- If we approximate ego vehicle dynamics to a control affine function $\dot{x_0} = f(x_0) + Bu$, the control safe set can be written as:
  - $U_S(\mathrm{t}) = \{u(t): L(t)u(t) \leq S(t) \text{ if } \phi \geq 0\}$
  - where $L(t) = \frac{\partial \phi}{\partial x_0} B, S(t) = \eta - \frac{\partial \phi}{\partial x_j} \dot{x}_j - \frac{\partial \phi}{\partial x_0} f$
  - Derivation: $\dot{\phi}(x) = \frac{\partial \phi}{\partial x_0} \dot{x}_0 + \frac{\partial \phi}{\partial x_j} \dot{x}_j = \frac{\partial \phi}{\partial x_0} (f + Bu) + \frac{\partial \phi}{\partial x_j} \dot{x}_j \leq \eta$
- If there are multiple surrounding objects, we can calculate the intersection of the control safe set for each object, which is a convex polytope, denoted as $U_S(\mathrm{t})$.

Paper contains typos:

$$U_S(t) = \{u(t) : L(t)\, u(t) \leq S(t) \text{ if } \phi \geq 0\}$$

where $L(t) = \frac{\partial x_0}{\partial x_j} B$ and $S(t) = -\eta - \frac{\partial \phi}{\partial x_j} \dot{x}_j - \frac{\partial \phi}{\partial x_0} f$, x_0 and x_j are the states of the ego and surrounding vehicle, respectively.

Not covered in exam

Safe Control Command

- Control command $u(t) = \begin{bmatrix} a_t \\ \delta_t \end{bmatrix}$ from the trajectory tracking controller may be unsafe. The safety controller maps it into the control safe set U_S by solving Quadratic Programming (QP) problem, to stay in U_S while minimizing its deviation from the potentially unsafe $u(t)$:
  - $u^*(t) = \text{argmin}_{u \in U_S} \frac{1}{2} [u - u(t)]^T W [u - u(t)]$
  - where W is a 2×2 weight matrix
- We thus obtain the modified safe control command $u^*(t) = \begin{bmatrix} a_t^* \\ \delta_t^* \end{bmatrix}$.

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

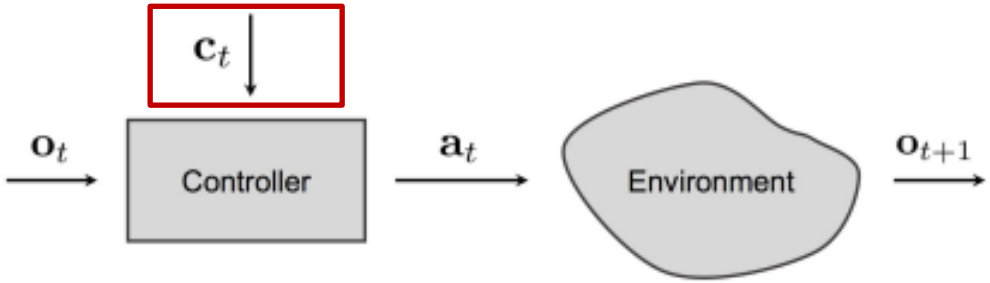


Fig. 2. High-level overview. The controller receives an observation $\mathbf{o}_t$ from the environment and a command $\mathbf{c}_t$. It produces an action $\mathbf{a}_t$ that affects the environment, advancing to the next time step.

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.

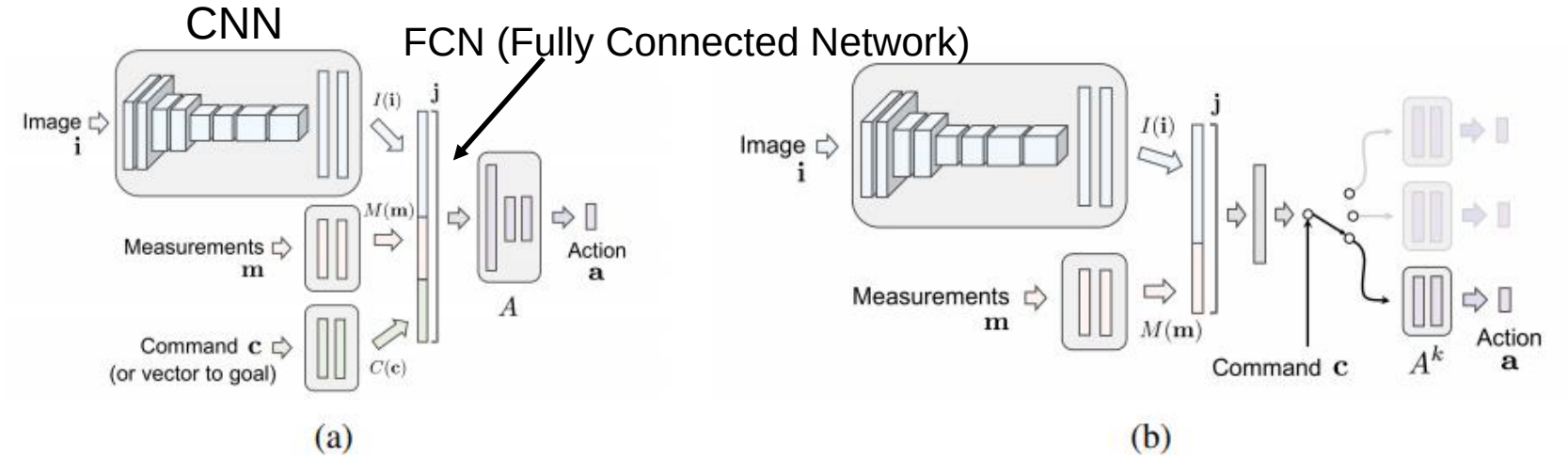


Fig. 3. Two network architectures for command-conditional imitation learning. (a) `command input`: the command is processed as input by the network, together with the image and the measurements. The same architecture can be used for goal-conditional learning (one of the baselines in our experiments), 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

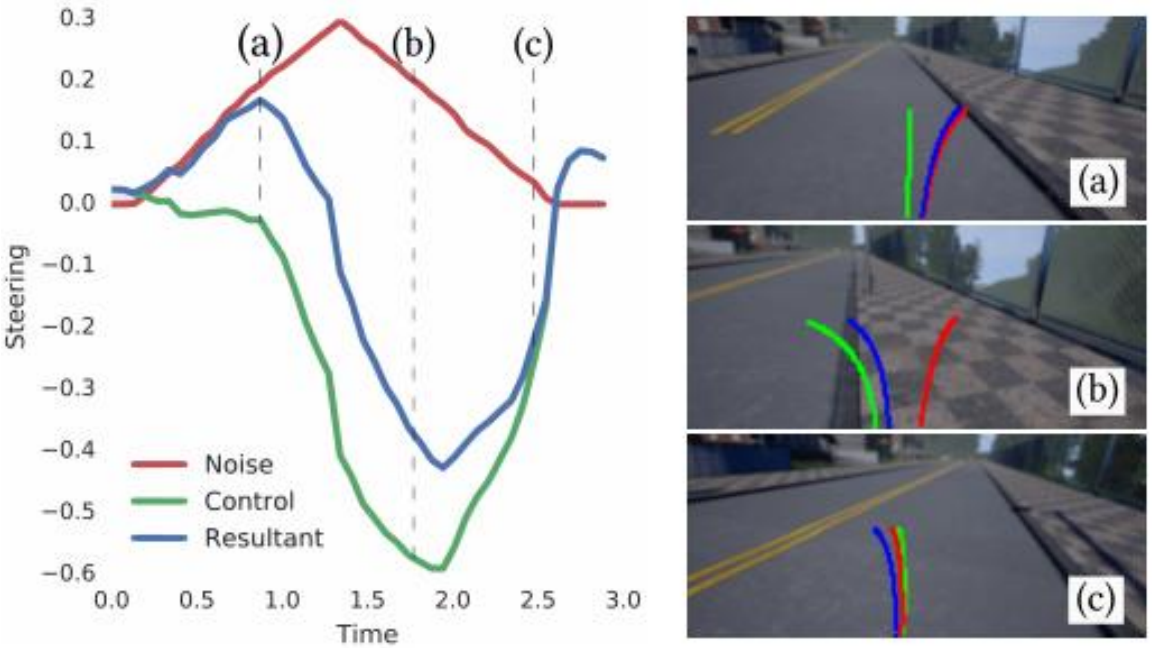


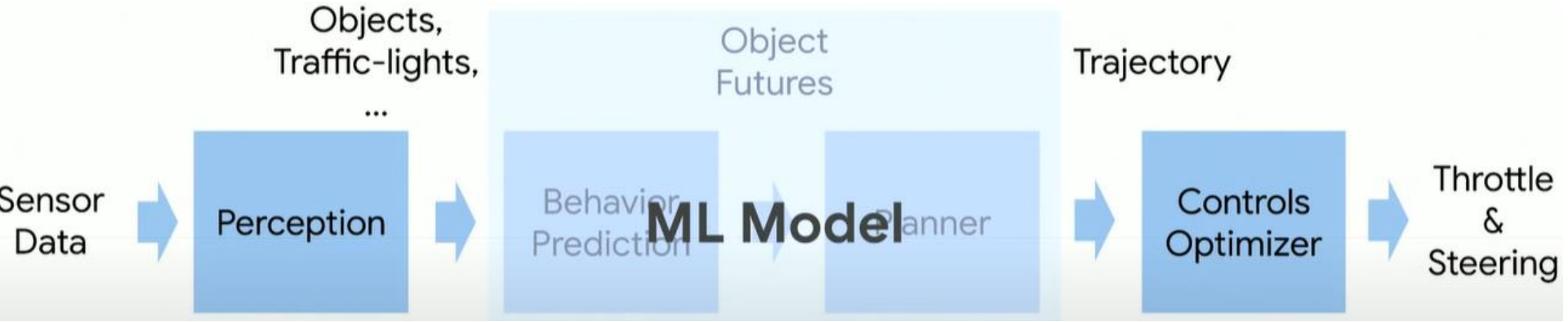
Fig. 4. Noise injection during data collection. We show a fragment from an actual driving sequence from the training set. The plot on the left shows steering control [rad] versus time [s]. In the plot, the red curve is an injected triangular noise signal, the green curve is the driver's steering signal, 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 and IL with Safety [Chen19] use 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 and IL with Safety are mid-to-mid (DNN used for behavior prediction and planning to generate trajectory).
- DNN Architecture
  - PilotNet, CIL and IL with Safety use CNN, input is current video frame only, with no history info.
    - I find it strange, as you can input a past history of N frames to capture recent history.
  - ChauffeurNet uses Convolutional RNN with an explicit memory, taking into account recent history.
- CIL : adds a condition variable to help with the decision process at intersections.
- IL with Safety has a safety controller that limits final control command to safe set.

[Müller 2018]

- Müller, Matthias, et al. "Driving policy transfer via modularity and abstraction." *arXiv preprint arXiv:1804.09364* (2018). (KAUST, Intel Labs)



Mid-to-Mid

Abstract

- End-to-end approaches to autonomous driving have high sample complexity and are difficult to scale to realistic urban driving. Simulation can help end-to-end driving systems by providing a cheap, safe, and diverse training environment. Yet training driving policies in simulation brings up the problem of transferring such policies to the real world. We present an approach to transferring driving policies from simulation to reality via modularity and abstraction. Our approach is inspired by classic driving systems and aims to combine the benefits of modular architectures and end-to-end deep learning approaches. The key idea is to encapsulate the driving policy such that it is not directly exposed to raw perceptual input or low-level vehicle dynamics. We evaluate the presented approach in simulated urban environments and in the real world. In particular, we transfer a driving policy trained in simulation to a 1/5-scale robotic truck that is deployed in a variety of conditions, with no finetuning, on two continents.

Modular Architecture

- The perception module takes as input a raw RGB image and outputs a segmentation map.
- The driving policy then takes this segmentation as input and produces waypoints indicating the desired local trajectory of the vehicle.
- The low-level controller, given the waypoints, generates the controls: steering angle and throttle.
- Driving Policy Transfer via Modularity and Abstraction
  - https://www.youtube.com/watch?v=BrMDJqI6H5U

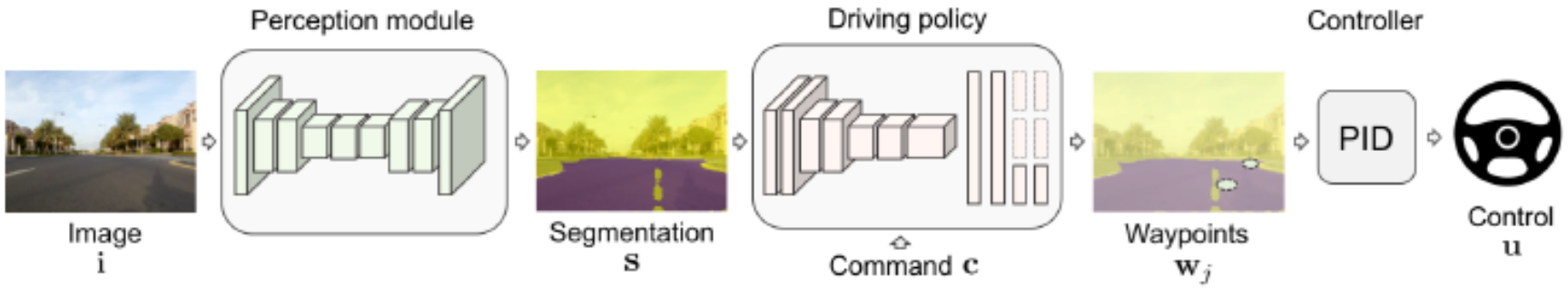


Figure 1: System architecture. The autonomous driving system comprises three modules: a perception module implemented by an encoder-decoder network, a command-conditional driving policy implemented by a branched convolutional network, and a low-level PID controller.

Perception Module for Image Segmentation

- We use a per-pixel binary segmentation of the image into “road” and “not road” regions. It abstracts away texture, lighting, shading, and weather, leaving only a few factors of variation: the geometry of the road, the camera pose, and the shape of objects occluding the road. Such segmentation contains sufficient information for following the road and taking turns, but it is abstract enough to support transfer.
- Generalization to new environments (e.g., different weather, country, etc.) or transfer to new domains (e.g., simulation to physical world) can be achieved by appropriately tuning the perception module.

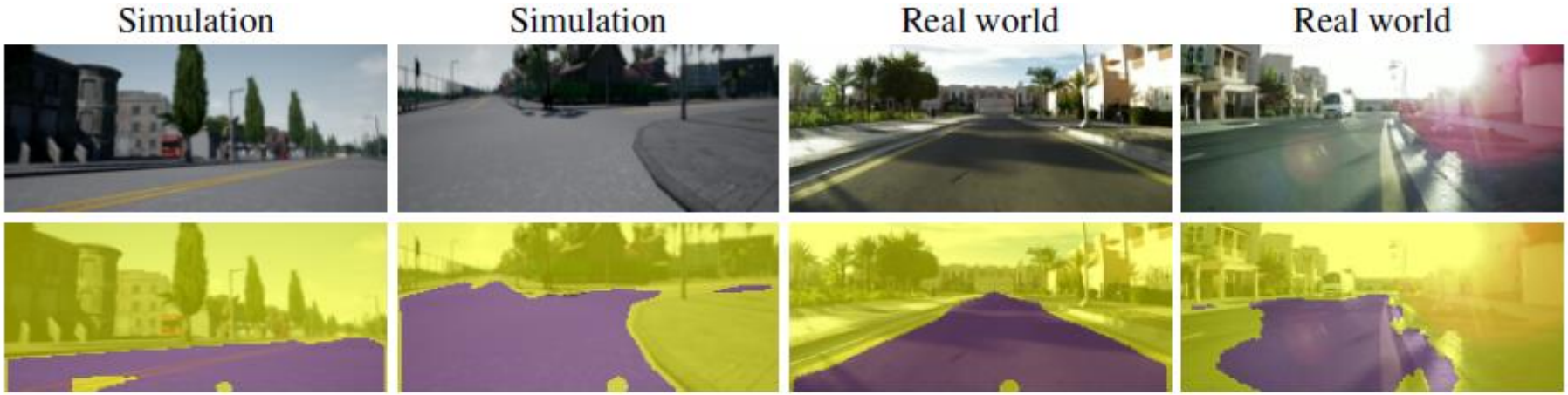


Figure 7: Sample outputs of the segmentation network trained on Cityscapes and tested in simulation and in the real world. The images are at the resolution used by the network – 200×88 pixels. The network works well in typical scenes both in simulation and in the real world, but accuracy drops under complex lighting conditions and in unusual situations.

Driving Policy Module

- Driving policy trained with Conditional IL [Codevilla 2018].
- At every frame, we predict two waypoints. One would be sufficient to control steering, but the second can be useful for longer-term maneuvers, such as controlling the throttle ahead of a turn. The waypoints w_j are encoded by the distance r_j and relative angle ϕ_j with respect to the heading direction:
  - $\phi_j = \angle(w_j, v), r_j = \|w_j\|$
  - We fix $r_1 = 5, r_2 = 20$, and only predict angles ϕ_1, ϕ_2.

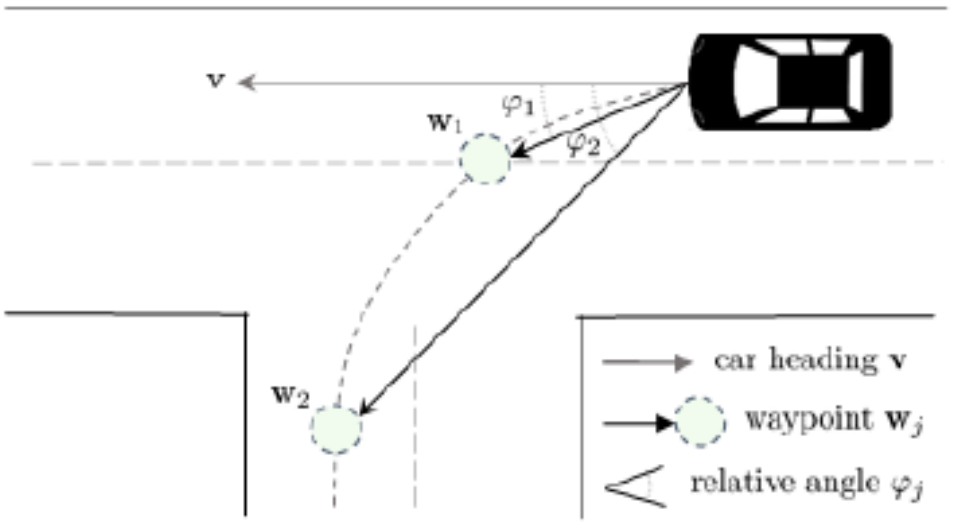


Figure 2: Waypoints are encoded by the distance to the vehicle and the relative angle to the vehicle's heading.

Loss Function for Training Driving Policy

- We start by collecting a training dataset $\{\langle o_i, c_i, a_i \rangle\}_{i=1}^{N}$ of observation-command-action tuples, from trajectories of an expert driving policy.
  - Observation o_i can be an image or a segmentation map;
  - Action a_i can be either vehicle controls (steering, throttle) or waypoints;
  - Command c_i is a categorical variable indicating one of three high-level navigation instructions (left, straight, right) at the next intersection.
- A function approximator f (a DNN) with learnable parameters θ is trained to predict actions from observations and commands: $\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_i l(a_i = f(o_i, c_i, \theta), a_{gt})$
  - L2 Loss function for each sample: $l(a_i, a_{gt}) = l(\langle s_i, a_i \rangle, \langle s_{gt}, a_{gt} \rangle) = \|s_i - s_{gt}\|^2 + \lambda_a \|a_i - a_{gt}\|^2$ (a_{gt} is ground action by expert).

The Expert Agent

- We program an expert agent to drive autonomously based on privileged information: precise map and location of the ego-vehicle.
- A global planner is used to randomly pick routes through a town and produce waypoints along the route. A PID controller is used to follow these waypoints.
- In order to increase the diversity of the dataset, the car is randomly initialized within the lane (not always in the center). In total, we record 28 hours of driving. To improve the robustness of the learned policy, we follow [Codevilla 2018] and introduce noise into the controls in approximately 20% of the data.

Data Augmentation

- We performing augmentation on the input images, not the segmentation maps
- When training segmentation networks we randomly perturb the input images:
  - Brightness, Saturation, Hue, Contrast
- When training driving policies, we randomly perturb the input images:
  - Gaussian blur, Additive Gaussian noise, Spatial dropout, Brightness additive, Brightness multiplicative, Contrast multiplicative, Saturation multiplicative…

Simulation Environment

Figure 3: Simulation environment. Maps of the two towns, along with example images that show the towns in two conditions: clear daytime (Weather 1) and cloudy daytime after rain (Weather 2). We use Town 1/Weather 1 during training. The other three combinations (Town 1/Weather 2, Town 2/Weather 1, and Town 2/Weather 2) are used to evaluate generalization in simulation. Note the significant visual differences between the towns and weather conditions.

Performance Evaluation

- img2ctrl: end-to-end
- img2wp: end-to-mid
- seg2ctrl: mid-to-end
- Ours: mid-to-mid

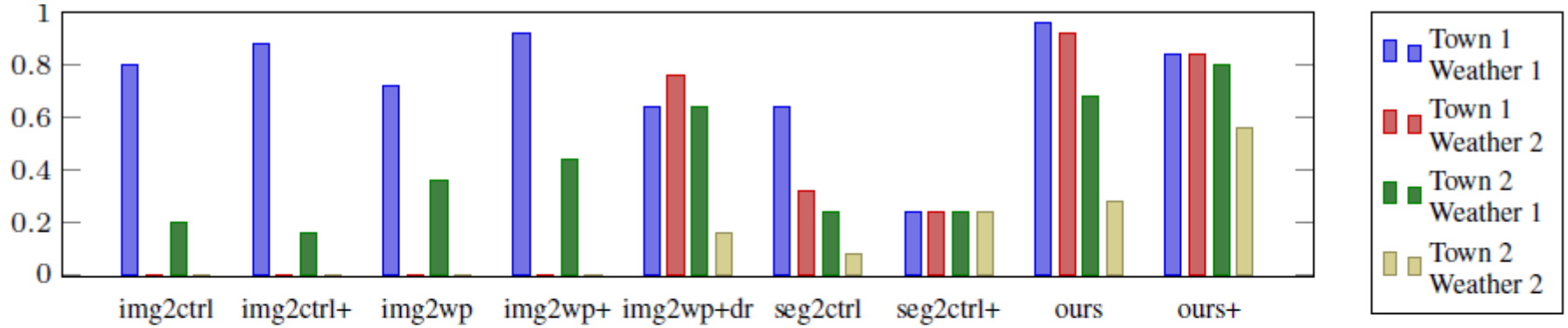
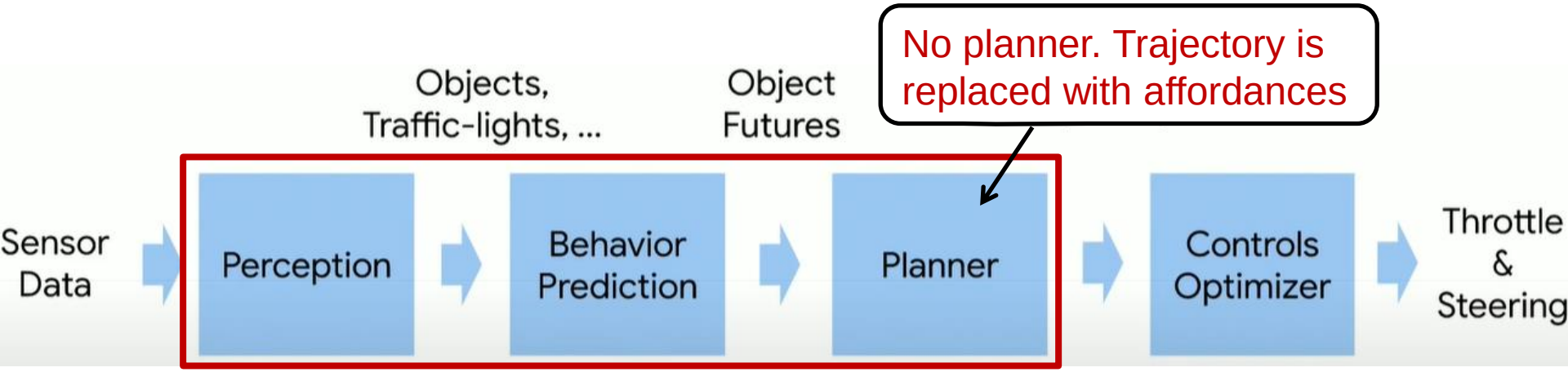


Figure 4: Quantitative evaluation of goal-directed navigation in simulation. We report the success rate over 25 navigation trials in four town-weather combinations. The models have been trained in Town 1 and Weather 1. The evaluated models are: *img2ctrl* – predicting low-level control from color images; *img2wp* – predicting waypoints from color images; *seg2ctrl* – predicting low-level control from the segmentation produced by the perception module; *ours* – predicting waypoints from the segmentation produced by the perception module. Suffix '+' denotes models trained with data augmentation, and '+dr' denotes the model trained with domain randomization.

Conditional Affordance Learning (CAL) [Sauer 2018]

- Sauer A, Savinov N, Geiger A. Conditional affordance learning for driving in urban environments[J]. arXiv preprint arXiv:1806.06498, 2018. (ETHZ, TU Munich and Max Planck Institute)



End-to-Mid

Abstract

- Most existing approaches to autonomous driving fall into one of two categories: modular pipelines, that build an extensive model of the environment, and imitation learning approaches, that map images directly to control outputs. A recently proposed third paradigm, direct perception, aims to combine the advantages of both by using a neural network to learn appropriate low-dimensional intermediate representations. However, existing direct perception approaches are restricted to simple highway situations, lacking the ability to navigate intersections, stop at traffic lights or respect speed limits. In this work, we propose a direct perception approach which maps video input to intermediate representations suitable for autonomous navigation in complex urban environments given high-level directional inputs. Compared to state-of-the-art reinforcement and conditional imitation learning approaches, we achieve an improvement of up to 68% in goal-directed navigation on the challenging CARLA simulation benchmark. In addition, our approach is the first to handle traffic lights and speed signs by using image-level labels only, as well as smooth car-following, resulting in a significant reduction of traffic accidents in simulation.

Conditional Affordance Learning (CAL)

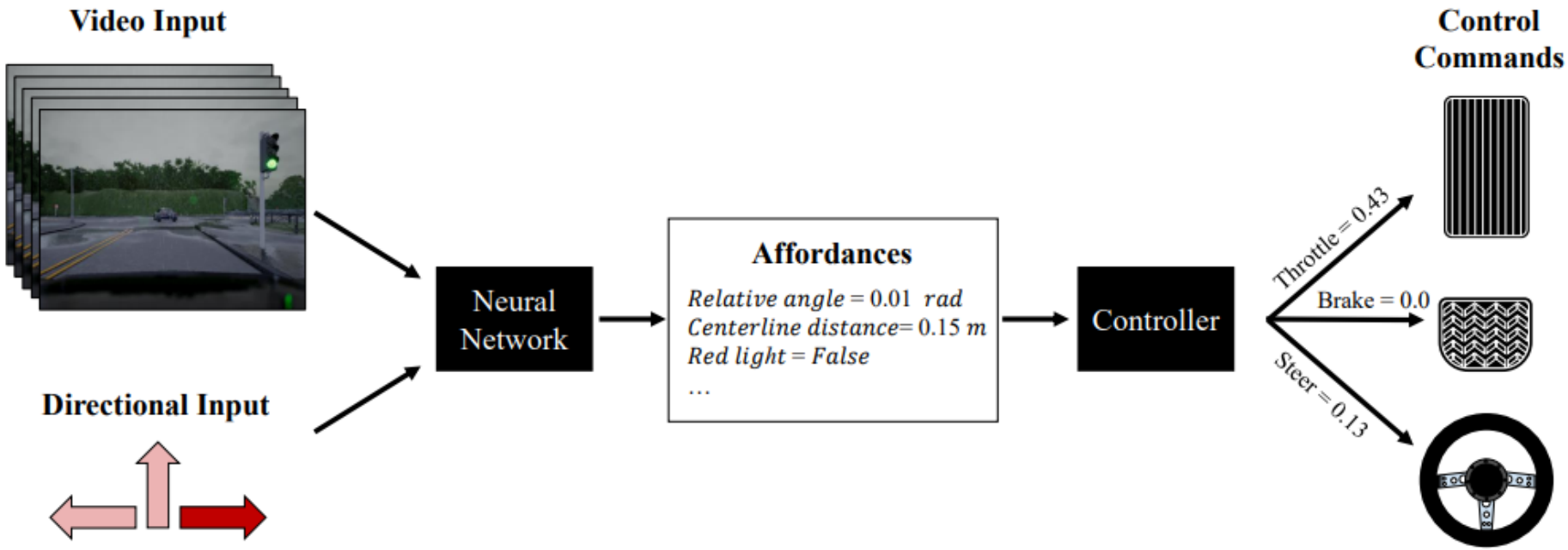


Figure 1: **Conditional Affordance Learning (CAL) for Autonomous Urban Driving.** The input video and the high-level directional input are fed into a neural network which predicts a set of affordances. These affordances are used by a controller to calculate the control commands.

Affordances

Type	Conditional	Affordances	Acronym	Range of values
discrete	No	Hazard stop	-	$\in \{True, False\}$
		Red Traffic Light	-	$\in \{True, False\}$
		Speed Sign [km/h]	-	$\in \{None, 30, 60, 90\}$
continuous	No	Distance to vehicle [m]	ℓ	$\in [0, 50]$
continuous	Yes	Relative angle [rad]	ψ	$\in [-\pi, \pi]$
		Distance to centerline [m]	d	$\in [-2, 2]$

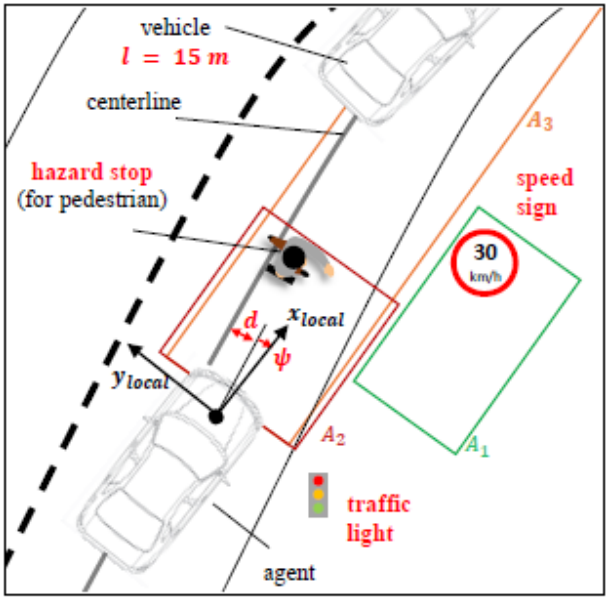
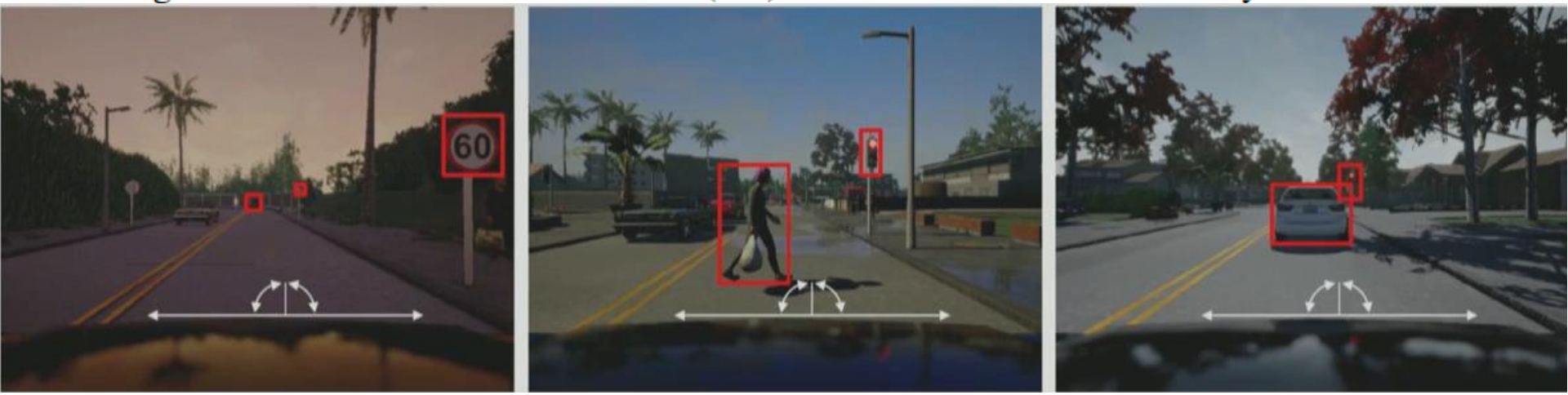


Figure 3: **Affordances.** Left: We categorize affordances according to their type (discrete/continuous) and whether they are conditional (dependent on directional input) or unconditional. Right: Illustration of the affordances (red) and observation areas used by our model.



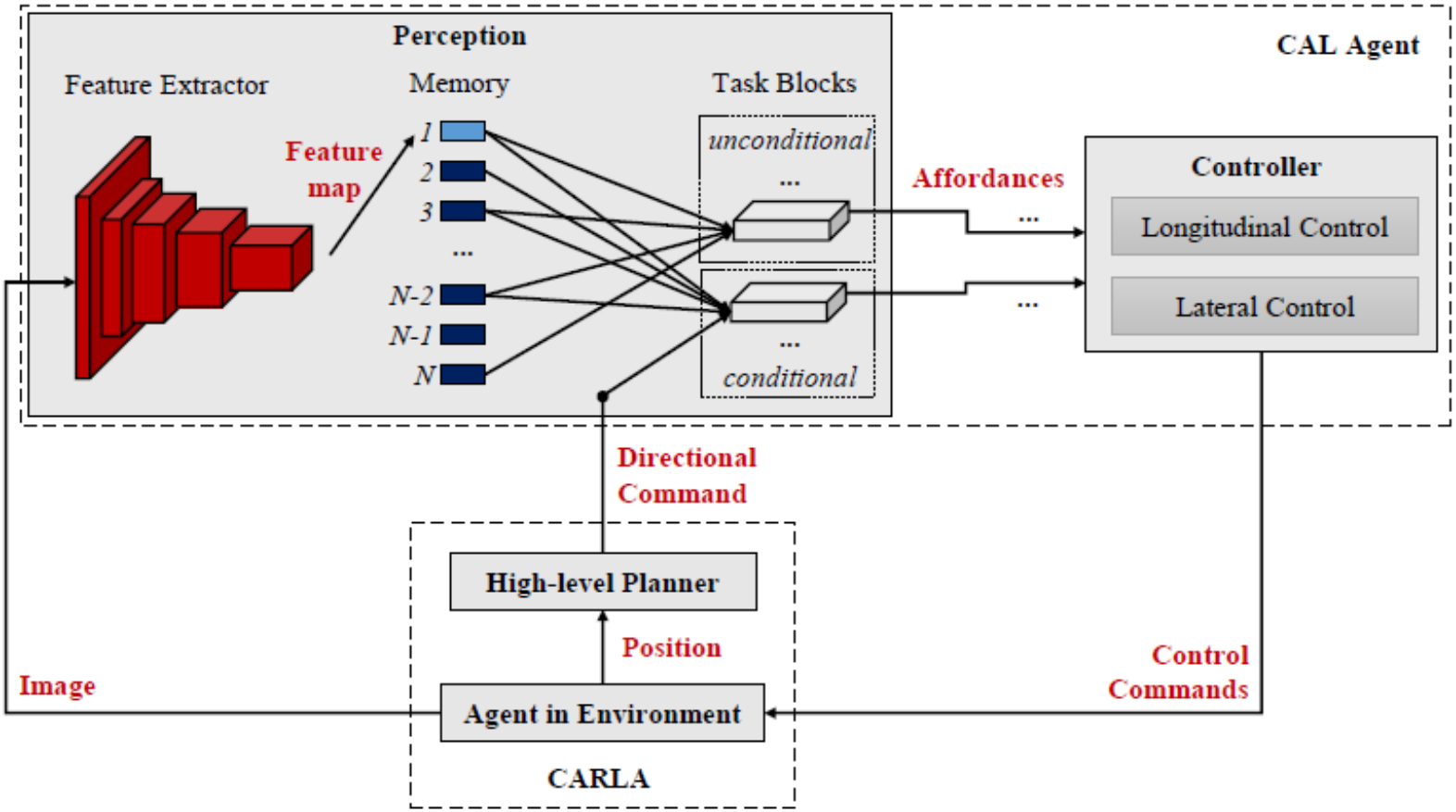
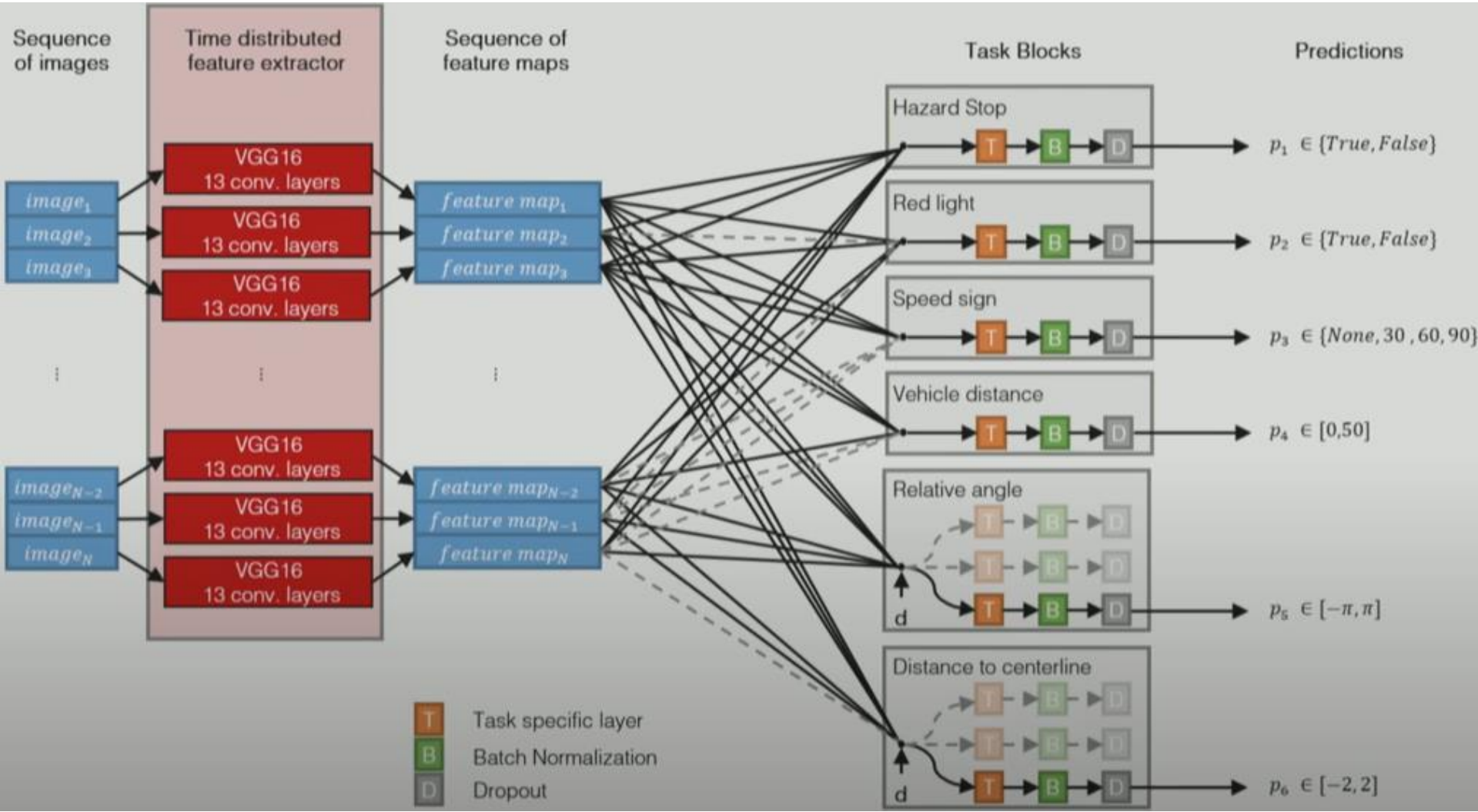


Figure 2: **Overview of our System.** The CAL agent (top) receives the current camera image and a directional command ("straight","left","right") from CARLA [27]. The feature extractor converts the image into a feature map. The agent stores the last N feature maps in memory, where N is the length of the input sequence required for the perception stack. This sequence of feature maps, together with the directional commands from the planner, are exploited by the task blocks to predict affordances. Different tasks utilize different temporal receptive fields and temporal dilation factors. The control commands calculated by the controller are sent back to CARLA which updates the environment and provides the next observation and directional command.

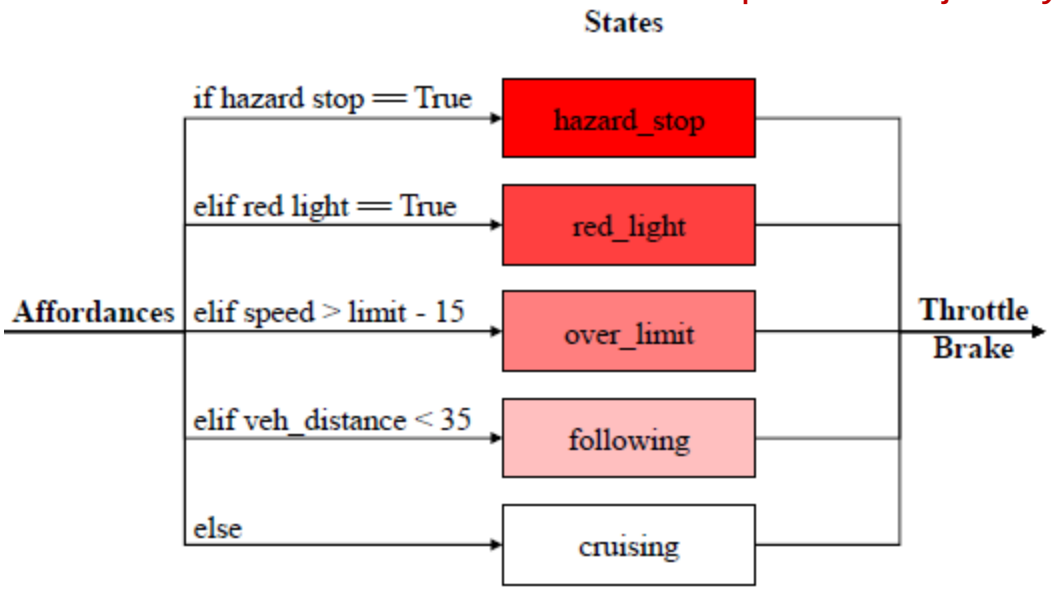
Perception DNN

- Loss function $\mathcal{L} = \sum_{j=1}^{3} H_j + \sum_{k=1}^{3} MAE_k$
  - H_j: Cross-Entropy Loss for classification;
  - MAE_k: Mean Absolute Error for regression
- Condition d acts as a switch in task blocks for “relative angle” and “distance to centerline”.



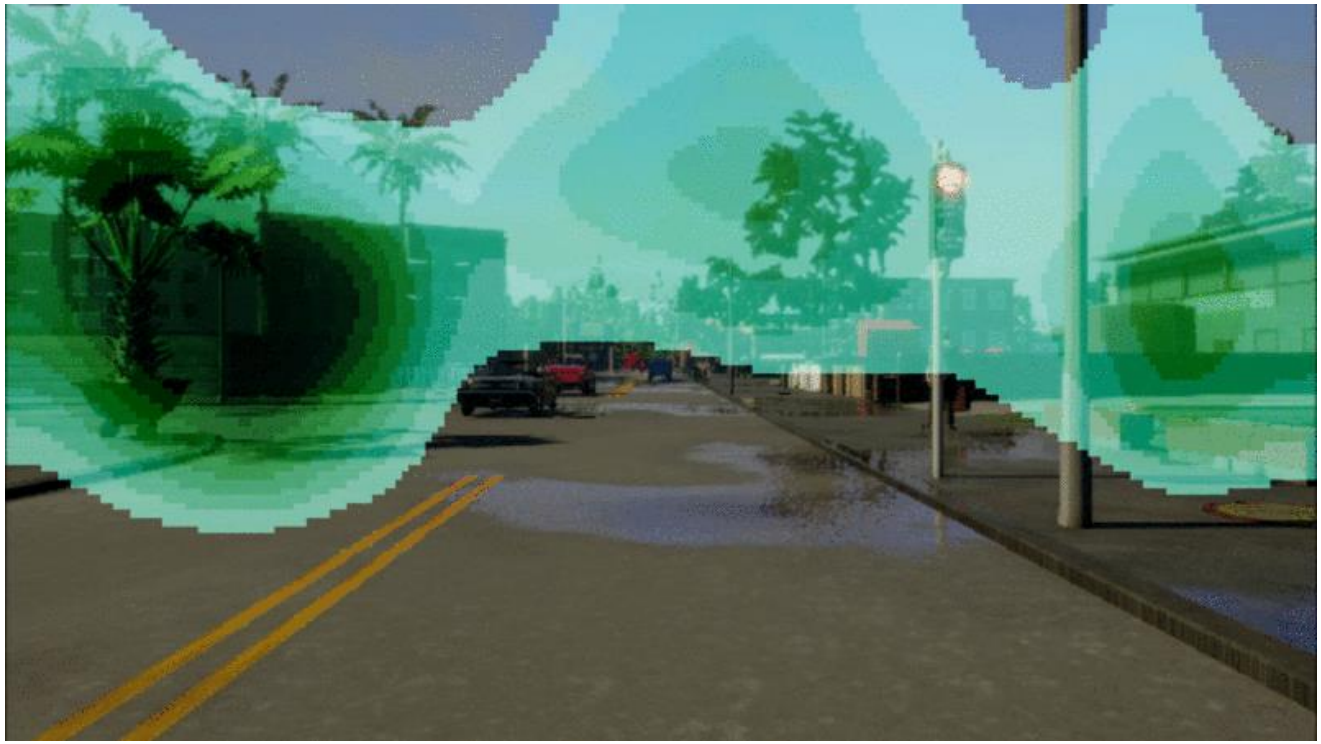
Longitudinal Control

- The states are ordered in descending importance from top-to-bottom as indicated by the color intensity. Control laws in each state:
- over_limit: $brake = .3 \cdot \frac{v(t)}{v^*(t)}$
  - $v(t)$: current speed; $v^*(t)$: speed limit
- red_light: $brake = .2 \cdot \frac{v(t)}{30}$
  - We use smaller multiplier .2 to slow down gradually; 30km/h is the typical speed zone where red lights are located.
- hazard_stop: $brake = 1$ (full braking)
- following and cruising: PID controller tuned with Ziegler-Nichols.
- Lateral control: standard Stanley controller (omitted)
- My thoughts: control based on affordances instead of a planned trajectory (waypoints).



Visualization

- We visualized which parts of the image the network attends to when predicting a specific affordance. We found that the network “looks” at the relevant parts of the image similar to what a human would do.
- e.g., when looking for a red light it observes the sides, where traffic lights are usually located. (Green color indicates no red light.)
- It recognizes crossing pedestrians as obstacles even before they appear in the input image by observing their shadows.



Performance Evaluation

- MP: Modular Pipeline
- CIL: Conditional IL [Codevilla 2018]
- RL: A3C [Mnih 2016]

Table 1: **Quantitative Evaluation on Goal-Directed Navigation Tasks.** We show the percentage of successfully completed episodes per task and compare CAL to a Modular Pipeline (MP) [29], Conditional Imitation Learning (CIL) [21] and Reinforcement Learning (RL) [29].

	Training conditions				New weather				New town				New town and new weather			
Task	MP	CIL	RL	CAL	MP	CIL	RL	CAL	MP	CIL	RL	CAL	MP	CIL	RL	CAL
Straight	98	95	89	**100**	**100**	98	86	**100**	92	**97**	74	93	50	80	68	**94**
One turn	82	89	34	**97**	95	90	16	**96**	61	59	12	**82**	50	48	20	**72**
Navigation	80	86	14	**92**	**94**	84	2	90	24	40	3	**70**	47	44	6	**68**
Nav. dynamic	77	**83**	7	**83**	**89**	82	2	82	24	38	2	**64**	44	42	4	**64**

CILRS [Codevilla 2019]

- Codevilla F, Santana E, López A M, et al. Exploring the limitations of behavior cloning for autonomous driving[C]//Proceedings of the IEEE International Conference on Computer Vision. 2019: 9329-9338. (U Barcelona, KAUST, Intel Labs)

Abstract

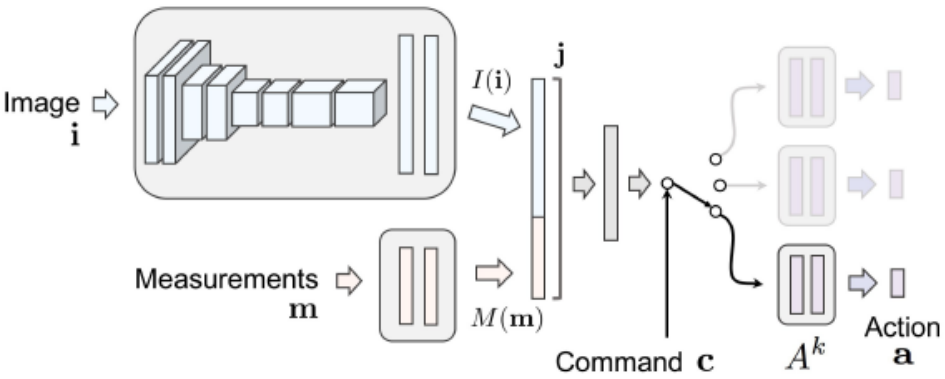
- we propose a new benchmark to experimentally investigate the scalability and limitations of behavior cloning. We show that behavior cloning leads to state-of-the-art results, including in unseen environments, executing complex lateral and longitudinal maneuvers without these reactions being explicitly programmed. However, we confirm well-known limitations (due to dataset bias and overfitting), new generalization issues (due to dynamic objects and the lack of a causal model), and training instability requiring further research before behavior cloning can graduate to real-world driving.

Limitations

- Bias in Naturalistic Driving Datasets
  - Most of real-world driving consists in either a few simple behaviors or a heavy tail of complex reactions to rare events.
- Causal Confusion
  - Spurious correlations cannot be distinguished from true causes in observed training demonstration pattern.
  - The inertia problem: When the ego vehicle is stopped (e.g., at a red traffic light), the probability it stays static is overwhelming in the training data. This creates a spurious correlation between low speed and no acceleration, inducing excessive stopping and difficult restarting in the imitative policy. Although mediated perception approaches that explicitly model causal signals like traffic lights do not suffer from this theoretical limitation, they still under-perform end-to-end learning in unconstrained environments, because not all causes might be modeled (e.g., some potential obstacles) and errors at the perception layer (e.g., missed detections) are irrecoverable.
- High variance
  - With a fixed off-policy training dataset, one would expect CIL to always learn the same policy in different runs of the training phase. However, the cost function is optimized via Stochastic Gradient Descent (SGD), which assumes the data is (i.i.d independent and identically distributed). When training a reactive policy on snapshots of longer human demonstrations included in the training data, the i.i.d. assumption does not hold. Consequently, we might observe a high sensitivity to the initialization and the order in which the samples are seen during training.

CILRS

- We propose a robustified CIL model designed to improve on [Codevilla 2018]
  - Deeper Residual Architecture (ResNet34)
  - Speed Prediction Regularization: jointly train a sensorimotor controller that predicts action a, with a network that predicts the ego vehicle's speed v_p. this joint optimization enforces the perception module to put speed-related features (e.g., traffic lights) into the learned representation, and alleviates the Causal Confusion problem.
  - (e.g., free space, curves, traffic light states, etc).
  - Use L1 Loss instead of L2 Loss, as it is more correlated to driving performance
  - Collect demonstrations from an expert game AI using privileged information to drive correctly (i.e. always respecting rules of t



Original Network [Codevilla 2018]

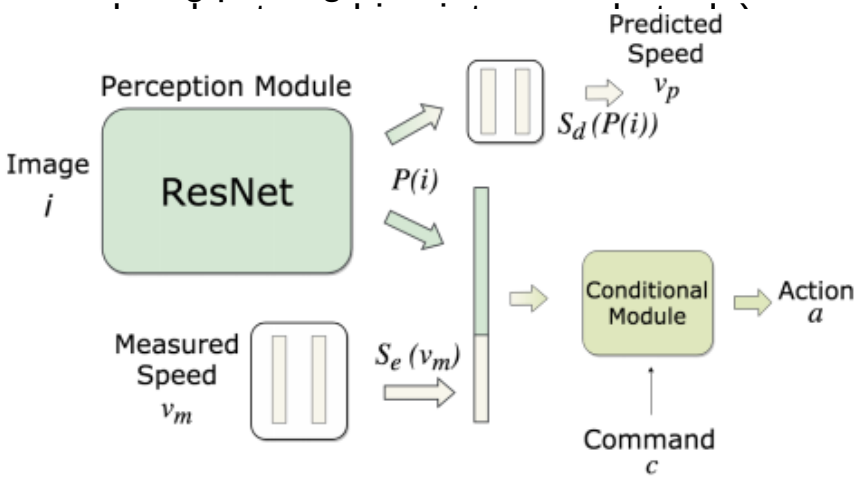


Figure 2. Our proposed network architecture, called CILRS, for end-to-end urban driving based on CIL [10]. A ResNet perception module processes an input image to a latent space followed by two prediction heads: one for controls and one for speed.

NoCrash Benchmark

- More tasks and metrics than the original CARLA benchmark.
- We propose three different tasks, each one corresponding to 25 goal directed episodes. In each episode, the agent starts at a random position and is directed by a high-level planner into reaching some goal position. The three tasks have the same set of start and end positions, as well as an increasing level of difficulty:
  - Empty Town
  - Regular Traffic
  - Dense Traffic
- We end the episode as failing when any collision bigger than a fixed magnitude happens

Performance Evaluation

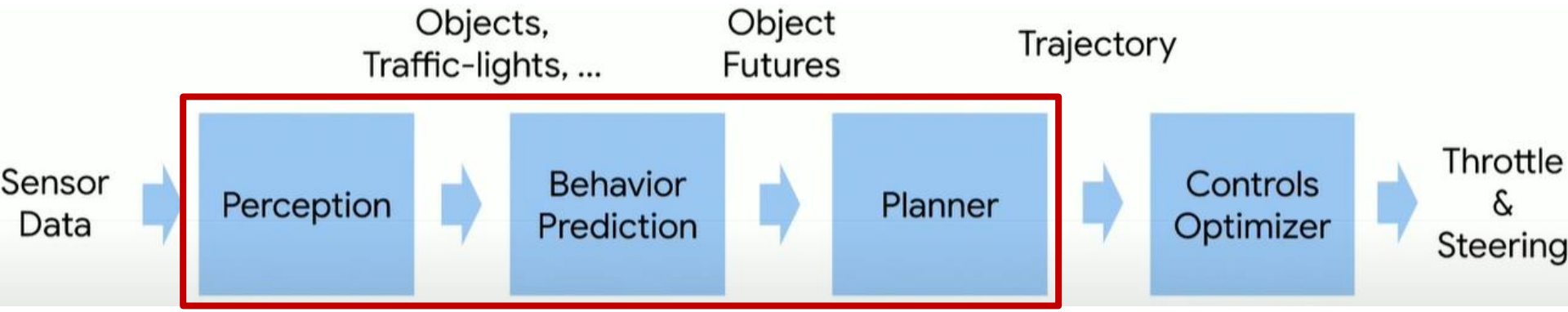
- CIL: Conditional IL [Codevilla 2018]
- CAL: Conditional Affordance Learning [Sauer 2018]
- MT: Multitask [Li 2018] (not discussed)
- CILR: CIL with ResNet but no Speed prediction
- CILRS: CIL with both ResNet and Speed prediction

	Training conditions					New Town & Weather				
Task	CIL[10]	CAL[36]	MT[25]	CILR	CILRS	CIL[10]	CAL[36]	MT[25]	CILR	CILRS
Empty	79 ± 1	81 ± 1	84 ± 1	92 ± 1	$\mathbf{97} \pm 2$	24 ± 1	25 ± 3	57 ± 0	66 ± 2	$\mathbf{90} \pm 2$
Regular	60 ± 1	73 ± 2	54 ± 2	72 ± 5	$\mathbf{83} \pm 0$	13 ± 2	14 ± 2	32 ± 2	54 ± 2	$\mathbf{56} \pm 2$
Dense	21 ± 2	$\mathbf{42} \pm 3$	13 ± 4	28 ± 1	$\mathbf{42} \pm 2$	2 ± 0	10 ± 0	14 ± 2	13 ± 4	$\mathbf{24} \pm 8$

Table 2. Results on our *NoCrash* benchmark. Mean and standard deviation on three runs, as CARLA 0.8.4 has significant non-determinism.

Learning by Cheating (LBC) [Chen 2019]

- Chen D, Zhou B, Koltun V, et al. Learning by cheating[J]. arXiv preprint arXiv:1912.12294, 2019. (UT Austin, Intel Labs)

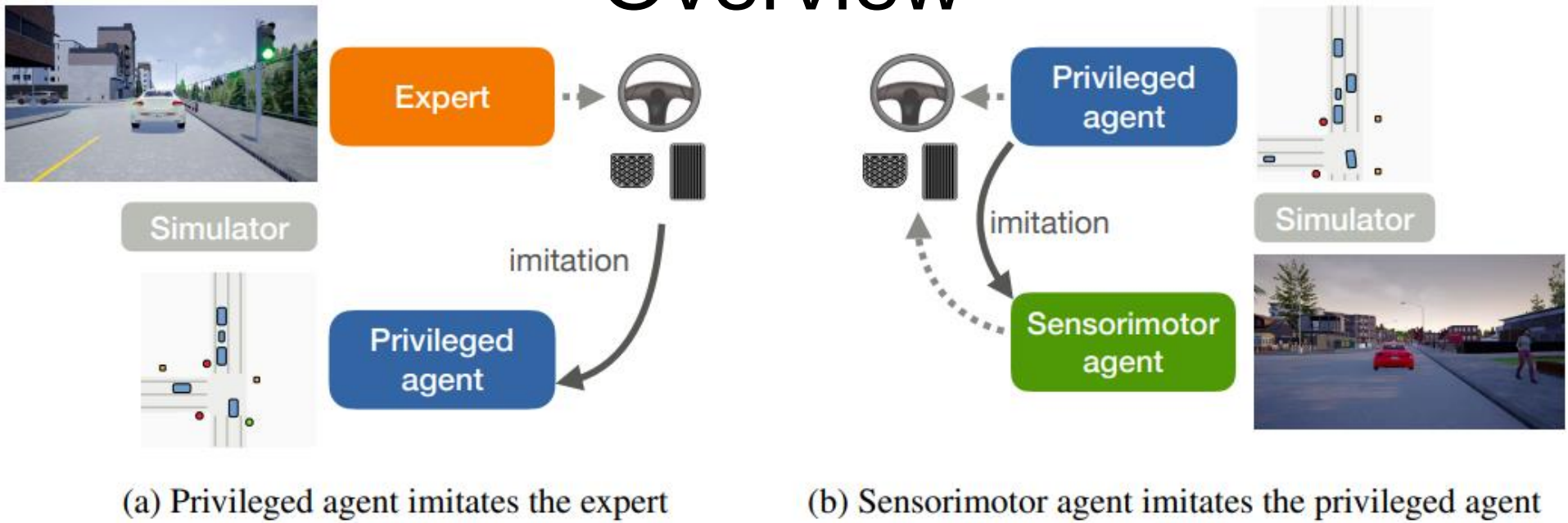


End-to-Mid

Abstract

- Vision-based urban driving is hard. The autonomous system needs to learn to perceive the world and act in it. We show that this challenging learning problem can be simplified by decomposing it into two stages.
- We first train an agent that has access to privileged information. This privileged agent cheats by observing the ground-truth layout of the environment and the positions of all traffic participants.
- In the second stage, the privileged agent acts as a teacher that trains a purely vision-based sensorimotor agent. The resulting sensorimotor agent does not have access to any privileged information and does not cheat.
- This two-stage training procedure is counter-intuitive at first, but has a number of important advantages that we analyze and empirically demonstrate. We use the presented approach to train a vision-based autonomous driving system that substantially outperforms the state of the art on the CARLA benchmark and the recent NoCrash benchmark. Our approach achieves, for the first time, 100% success rate on all tasks in the original CARLA benchmark, sets a new record on the NoCrash benchmark, and reduces the frequency of infractions by an order of magnitude compared to the prior state of the art.
- Learning by Cheating
  - https://www.youtube.com/watch?v=u9ZCxxD-UUw

Overview



(a) Privileged agent imitates the expert (b) Sensorimotor agent imitates the privileged agent

Figure 1: Overview of our approach. **(a)** An agent with access to privileged information learns to imitate expert demonstrations. This agent learns a robust policy by cheating. It does not need to learn to see because it gets direct access to the environment's state. **(b)** A sensorimotor agent without access to privileged information then learns to imitate the privileged agent. The privileged agent is a "white box" and can provide high-capacity on-policy supervision. The resulting sensorimotor agent does not cheat.

- (a) Privileged agent imitates the expert (Behavior Cloning (off-policy))
- (b) Sensorimotor agent imitates the privileged agent (first Behavior Cloning (off-policy), then Direct Policy Learning (on-policy))

Why Does it Work?

- The effectiveness of this decomposition is counter-intuitive. If direct IL – from expert trajectories to vision-based driving – is hard, why is the decomposition of the learning process into two stages, both of which perform imitation, any better?
- First, the privileged agent operates on a compact intermediate representation of the environment, and can thus learn faster and generalize better. In particular, the representation we use (a bird's-eye view) enables simple and effective data augmentation that facilitates generalization.
- Second, the trained privileged agent can provide much stronger supervision than the original expert trajectories. It can be queried from any state of the environment, not only states that were visited in the original trajectories. It turns passive expert trajectories into an online agent that can provide adaptive on-policy supervision.
- Third, if the privileged agent is trained via conditional imitation learning, it can provide an action for each possible command (e.g., "turn left", "turn right"), all at once, in any state of the environment. Thus all conditional branches of the privileged agent can train all branches of the sensorimotor agent in parallel. In every state visited during training, the sensorimotor student can in effect ask the privileged teacher "What would you do if you had to turn left here?", "What would you do if you had to turn right here?", etc.
- My thoughts: an analogy: expert is a Kung Fu master who lives in the mountains, provides recorded video lectures, but is not available for interactive instructions; privileged agent is a student with perfect eyesight (with bird's-eye view), hence can learn effectively from the master; sensorimotor agent is a young pupil with poor eyesight (with front-camera view), who can learn more effectively from the junior privileged agent with interactive instructions than from the master directly.

Agent Architecture

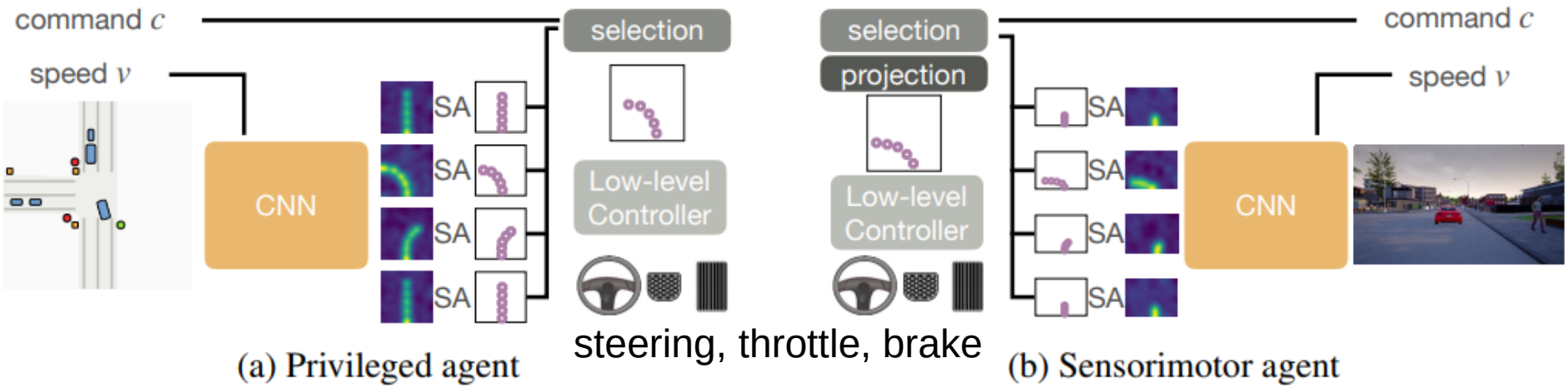


Figure 2: Agent architectures. **(a)** The privileged agent receives a bird's-eye view image of the environment and produces a set of heatmaps that go through a soft-argmax layer (SA), yielding waypoints for all commands. The input command selects one conditional branch. The corresponding waypoints are given to a low-level controller that outputs the steering, throttle, and brake. **(b)** The sensorimotor agent receives genuine sensory input (image from a forward-facing camera). It produces waypoints in the egocentric camera frame. Waypoints are selected based on the command, projected into the vehicle's coordinate frame, and passed to the low-level controller.

Loss Functins

- Privileged agent tries to imitate expert by minimizing expected difference between ground truth trajectory and predicted trajectory: $\min_{\theta} E_{(M,v,c,\boldsymbol{w})\sim\tau} \|\boldsymbol{w} - f_{\theta}^{*}(M,v)^{c}\|_{1}$
- Sensorimotor agent tries to imitate privileged agent by minimizing expected difference between own predicted trajectory and that by privileged agent : $\min_{\theta} E_{(M,I,v)\sim D} \left\| T_{p}(f(I,v)) - f_{\theta}^{*}(M,v) \right\|_{1}$
  - D is a dataset of corresponding road maps M, images I, and velocities v.
  - Sampling is no longer restricted to the offline trajectories provided by the original expert. In particular, the learning algorithm can sample states adaptively by rolling out the sensorimotor agent during training.
  - The sensorimotor agent can be supervised on all its waypoints and across all commands c at once.

Data Augmentation

(a) Road map (b) Rotation and shift aug.

Figure 3: **(a)** Map M provided to the privileged agent. One channel each for road (light grey), lane boundaries (grey), vehicles (blue), pedestrians (orange), and traffic lights (green, yellow, and red). The agent is centered at the bottom of the map. The agent's vehicle (dark red) and predicted waypoints (purple) are shown for visualization only and are not provided to the network. **(b)** The map representation affords simple and effective data augmentation via rotation and shifting.

Controller

- Both privileged and sensorimotor agents rely on low-level controller
  - Input: a set of waypoints $\hat{w} = \{\hat{w}_1, \ldots, \hat{w}_K\}$ in vehicle's coordinate frame
  - Output: steering, throttle, braking commands
- Longitudinal PID control tries to track target velocity $v_t^* = \frac{1}{K}\sum_{k=1}^{K} \frac{\|\hat{w}_i - \hat{w}_{i-1}\|_2}{\delta t}$
  - δt: temporal spacing between waypoints
  - https://en.wikipedia.org/wiki/Norm_(mathematics)
- Lateral PID controller tries to match a target heading angle towards $s^* = \tan^{-1}\frac{p_y}{p_x}$.
  - We first fit an arc to all waypoints and steer towards a point on the arc to average out prediction error in individual waypoints.
  - The point $p = (p_x, p_y)$ is a projection of one of the predicted waypoints onto the arc. We use w_2 for the straight and follow-the-road commands (shown in fig), w_3 for right turn, and w_4 for left turn. Later waypoints allow for a larger turning radius.
  - (My thoughts: Paper uses the term steering angle, but it is actually heading angle (in bicycle model.))

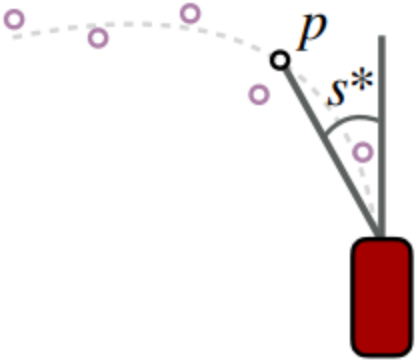


Figure 4: Lateral PID controller. Here the agent aims at the projection of the second waypoint onto the fitted arc. s^* denotes the angle between the vehicle and the target point p.

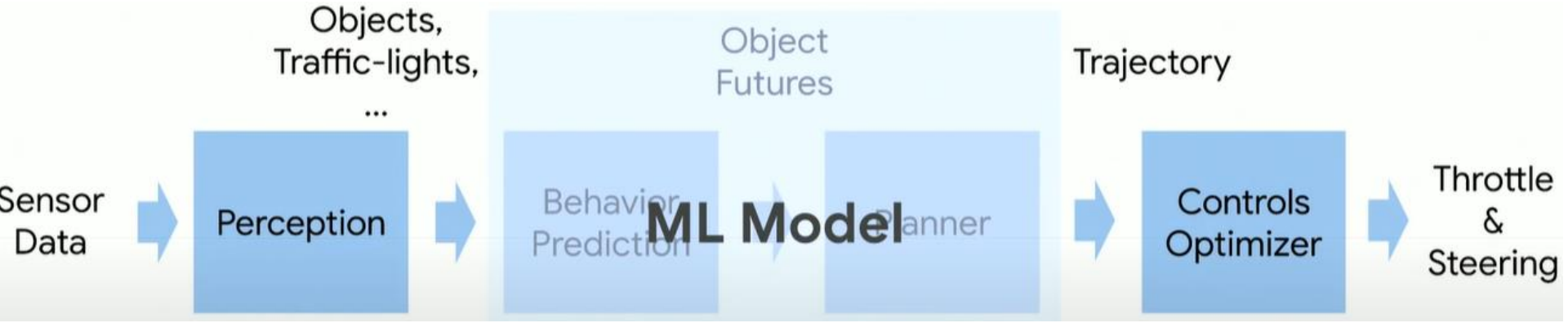
Performance Evaluation

Task	Weather	MP [8]	CIL [6]	CIRL [14]	CAL [22]	CILRS [7]	**LBC**	**LBC†**
Straight	train	92	97	**100**	93	96	**100**	**100**
One turn		61	59	71	82	84	**100**	**100**
Navigation		24	40	53	70	69	**100**	**98**
Nav. dynamic		24	38	41	64	66	**99**	**99**
Straight	test	50	80	98	94	96	**100**	**100**
One turn		50	48	80	72	92	**100**	**100**
Navigation		47	44	68	88	92	**100**	**100**
Nav. dynamic		44	42	62	64	90	**100**	**100**

Table 2: Comparison of the success rate of the presented approach (LBC) to the state of the art on the original CARLA benchmark (*CoRL2017*) in the test town. (The supplement provides results on the training town.) LBC† denotes our agent trained and evaluated on CARLA 0.9.6. All other agents were evaluated on CARLA 0.8 and 0.9.5. Our approach outperforms all prior work and achieves 100% success rate on all routes in the full-generalization setting (test town, test weather).

Deep Imitative Models (DIM) [Rhinehart 2020]

- Nicholas Rhinehart, Rowan McAllister, Sergey Levine, Deep Imitative Models for Flexible Inference, Planning, and Control, ICLR 2020



Mid-to-Mid

Abstract

- Imitation Learning (IL) is an appealing approach to learn desirable autonomous behavior. However, directing IL to achieve arbitrary goals is difficult. In contrast, planning-based algorithms use dynamics models and reward functions to achieve goals. Yet, reward functions that evoke desirable behavior are often difficult to specify. In this paper, we propose "Imitative Models" to combine the benefits of IL and goal-directed planning. Imitative Models are probabilistic predictive models of desirable behavior able to plan interpretable expert-like trajectories to achieve specified goals. We derive families of flexible goal objectives, including constrained goal regions, unconstrained goal sets, and energy-based goals. We show that our method can use these objectives to successfully direct behavior. Our method substantially outperforms six IL approaches and a planning-based approach in a dynamic simulated autonomous driving task, and is efficiently learned from expert demonstrations without online data collection. We also show our approach is robust to poorly specified goals, such as goals on the wrong side of the road.

DIM

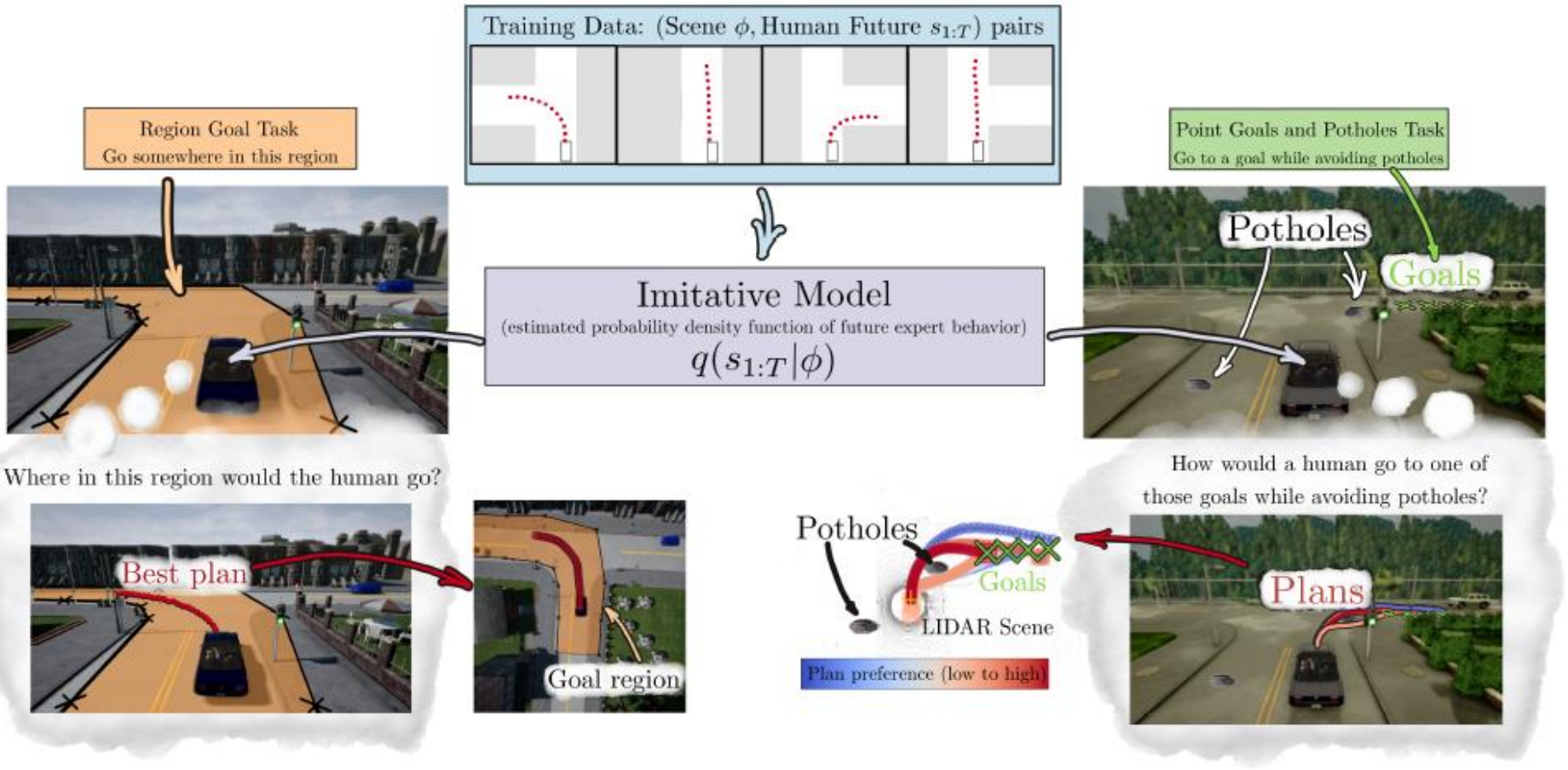


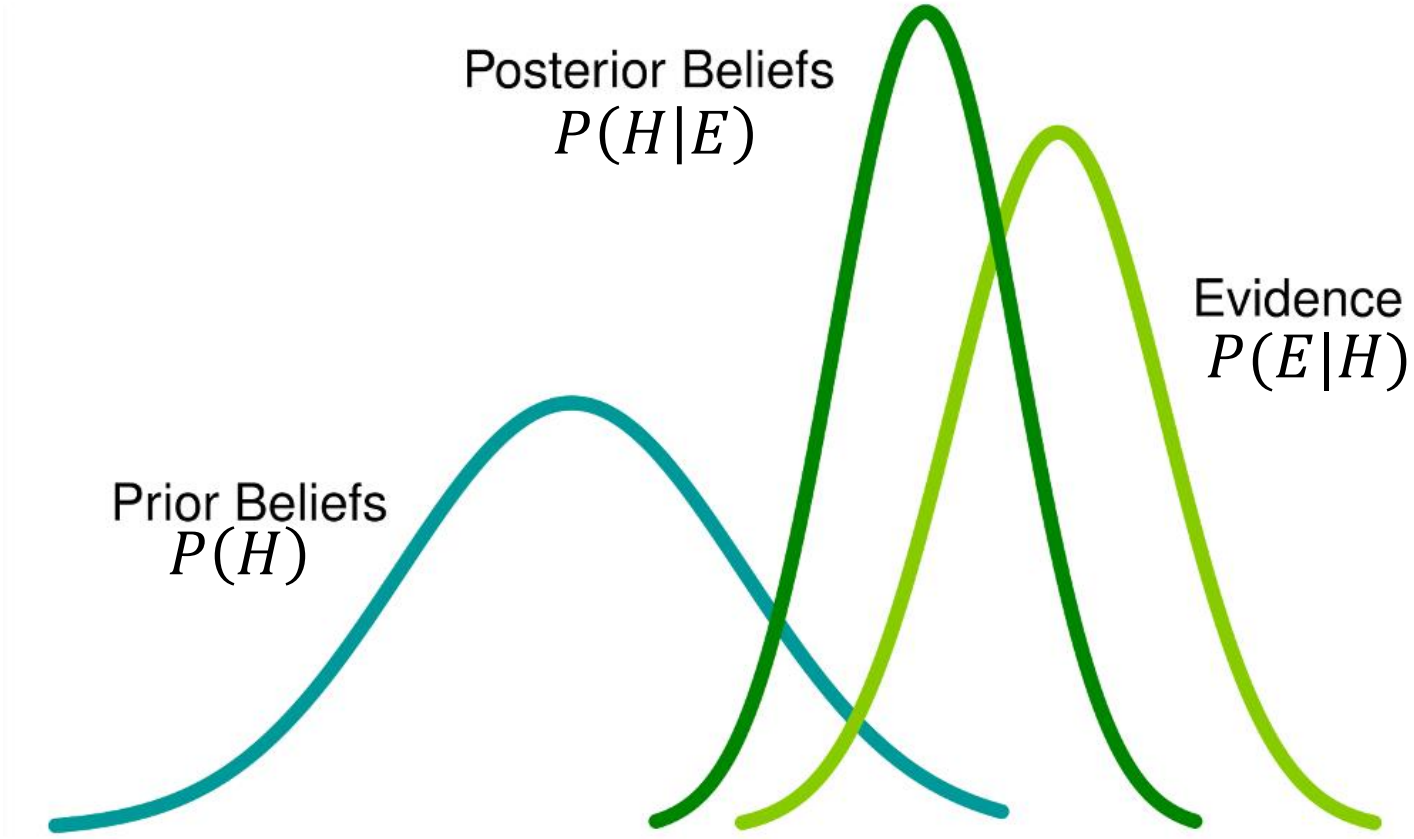
Figure 1: Our method: deep imitative models. *Top Center*. We use demonstrations to learn a probability density function q of future behavior and deploy it to accomplish various tasks. *Left*: A region in the ground plane is input to a planning procedure that reasons about how the expert would achieve that task. It coarsely specifies a destination, and guides the vehicle to turn left. *Right*: Goal positions and potholes yield a plan that avoids potholes and achieves one of the goals on the right.

DIM Details

Figure 5: Illustration of our method applied to autonomous driving. Our method trains an imitative model from a dataset of expert examples. After training, the model is repurposed as an imitative planner. At test-time, a route planner provides waypoints to the imitative planner, which computes expert-like paths to each goal. The best plan is chosen according to the planning objective and provided to a low-level PID-controller in order to produce steering and throttle actions. This procedure is also described with pseudocode in Appendix A.

Bayes Theorem

- $P(H|E) = \frac{P(E|H)P(H)}{P(E)}$
  - H: Hypothesis; E: Evidence
- 3Blue1Brown on Bayes theorem
  - https://www.youtube.com/watch?v=HZGCoVF3YvM&t=14s



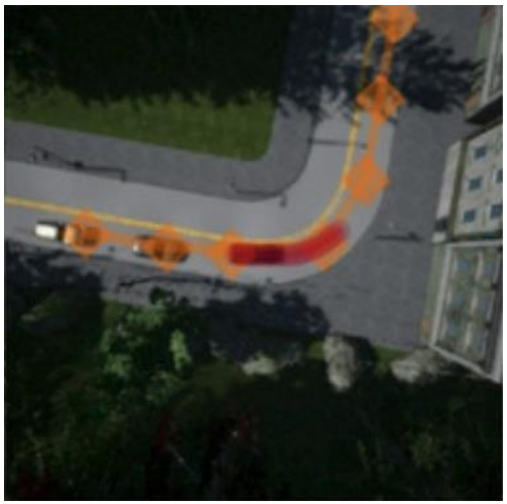
https://www.analyticsvidhya.com/blog/2016/06/bayesian-statistics-beginners-simple-english/

Planning as MAP (Maximum A Posteriori) Inference

- $s^*_{1:T} = \text{argmax}_{s_{1:T}} \log p(s_{1:T}|\mathcal{G}, \phi) = \text{argmax}_{s_{1:T}} \log q(s_{1:T}|\phi) + \log p(\mathcal{G}|s_{1:T}, \phi) - \log p(\mathcal{G}|\phi) = \text{argmax}_{s_{1:T}} \log q(s_{1:T}|\phi) + \log p(\mathcal{G}|s_{1:T}, \phi)$
- s_t: agent state at time t ($t = 0$ is current time step; T is planning horizon).
- ϕ: all of the agent's observations.
- $q(s_{1:T}|\phi)$: learned imitation prior. After training, $q(s_{1:T}|\phi)$ can generate trajectories that resemble those that the expert might generate – e.g. trajectories that navigate roads with expert-like maneuvers. However, these maneuvers will not have a specific goal. Beyond generating human-like behaviors, we wish to direct our agent to goals and have the agent automatically reason about the necessary mid-level details. We define general tasks by a set of goal variables $\mathcal{G}$.
- $p(\mathcal{G}|s_{1:T}, \phi)$: test-time goal likelihood. We give examples of p(Gjs;) after deriving a maximum a
- $p(s_{1:T}|\mathcal{G}, \phi)$: an expert-like plan that also tries to achieve goals. Probability of a plan $s_{1:T}$ conditioned on the goal G, modelled by a posterior distribution,

Goal Likelihood

- Constraint-based planning to goal sets: $p(\mathcal{G}|s_{1:T}) = \delta_{S_T}(\mathcal{G})$ delta function centered around goal (figures below)
  - $\mathcal{G} = \{g_1, g_2, \dots, g_k\}$ Way points (from high-level A* planner)
  - $\mathcal{G} = \{seg_1, seg_2, \dots, seg_k\}$ Line Segments
  - $\mathcal{G} = Polygon$ Region
- Unconstrained planning to goal sets: Gaussian distribution instead of delta function
  - $p(\mathcal{G}|s, \phi) \leftarrow \mathcal{N}(g_T; s_T, \epsilon T)$ Gaussian Final-State likelihood (reach a single goal state)
  - $p(\mathcal{G}|s, \phi) \leftarrow \prod_{k=T-K+1}^{T} \mathcal{N}(g_k; s_k, \epsilon T)$ Gaussian State Sequence (reach a sequence of K goal states)
  - $p(\mathcal{G}|s, \phi) \leftarrow \frac{1}{K} \prod_{k=1}^{K} \mathcal{N}(g_T^k; s_T, \epsilon T)$ Gaussian Final-State Mixture (reach any one of K goal states)
  - Goal likelihood encourages goals, rather than dictating them as for constraint-based. Useful for tolerating errors in goal-specification. Hyper-param ϵ controls variance.



Robustness to Errors in Goal-Specification.

- To test model's capability to stay in the distribution of demonstrated expert behavior, we designed a "decoy waypoints" experiment.
  - Navigating with high-variance waypoints.
    - Half of the waypoints are highly perturbed versions of the other half, serving as distractions for our Gaussian Final-State Mixture imitative planner.
  - Navigating with waypoints on the wrong side of the road.
- The planned path still stays on the road to mimic the expert behavior, rather than following the goals blindly, thanks to the learned imitation prior $q(s_{1:T}|\phi)$, which outweighs the influence of bad goals $p(\mathcal{G}|s_{1:T},\phi)$ when computing $s^*_{1:T}$, if $p(\mathcal{G}|s,\phi)$ has high variance (with large ϵ).

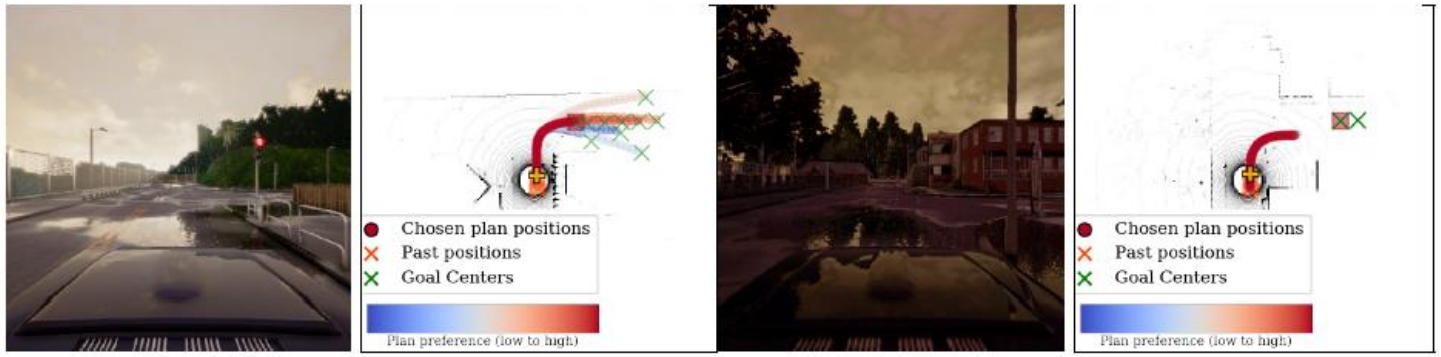


Figure 10: Tolerating bad goals. The planner prefers goals in the distribution of expert behavior (on the road at a reasonable distance). *Left*: Planning with $1/2$ decoy goals. *Right*: Planning with all goals on the wrong side of the road.

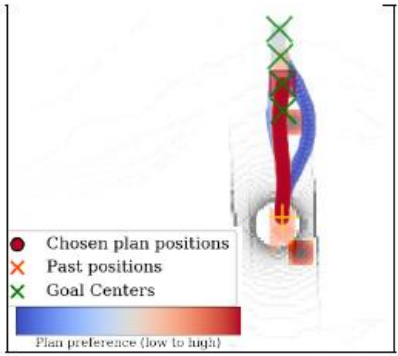


Figure 11: Test-time plans steering around potholes.

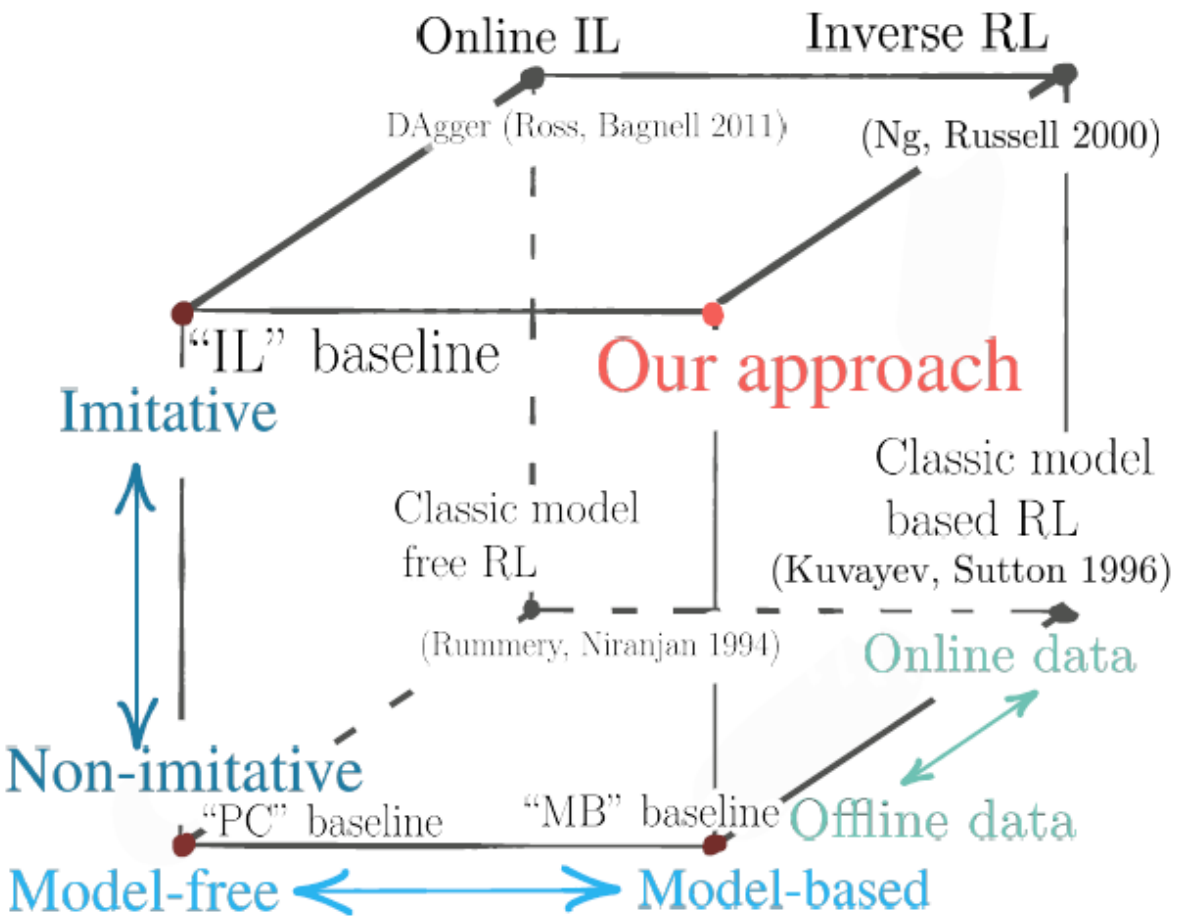


Figure 2: A brief taxonomy of learning-based control methods. In our scenario, we avoid online data collection, specifically from the policy we seek to imitate. We structure our imitation learner with a model to make it flexible to new tasks at test time. We compare against other offline approaches (front face).

Table 1: Desirable attributes of each approach. A green check denotes that a method has a desirable attribute, whereas a red cross denotes the opposite. A "†" indicates an approach we implemented.

Approach	Flexible to New Goals	Trains without goal labels	Outputs Plans	Trains Offline	Has Expert P.D.F.
CIRL* (Liang et al., 2018)	✗	✗	✗	✗	✗
CAL* (Sauer et al., 2018)	✗	✗	✗	✓	✗
MT* (Li et al., 2018)	✗	✗	✗	✓	✗
CIL* (Codevilla et al., 2018)	✗	✗	✗	✓	✗
CILRS* (Codevilla et al., 2019)	✗	✗	✗	✓	✗
CILS†	✗	✓	✗	✓	✗
MBRL†	✓	✓	✓	✗	✗
Imitative Models (*Ours*)†	✓	✓	✓	✓	✓

Table 2: Algorithmic components of each approach. A "†" indicates an approach we implemented.

Approach	Control Algorithm	← Learning Algorithm	← Goal-Generation Algorithm	← Routing Algorithm	High-Dim. Obs.
CIRL* (Liang et al., 2018)	Policy	Behavior Cloning+RL	Waypoint Classifier	A* Waypointer	Image
CAL* (Sauer et al., 2018)	PID	Affordance Learning	Waypoint Classifier	A* Waypointer	Image
MT* (Li et al., 2018)	Policy	Behavior Cloning	Waypoint Classifier	A* Waypointer	Image
CIL* (Codevilla et al., 2018)	Policy	Behavior Cloning	Waypoint Classifier	A* Waypointer	Image
CILRS* (Codevilla et al., 2019)	Policy	Behavior Cloning	Waypoint Classifier	A* Waypointer	Image
CILS†	PID	Trajectory Regressor	Waypoint Classifier	A* Waypointer	(LIDAR,λ)
MBRL†	Reachability Tree	State Regressor	Waypoint Selector	A* Waypointer	(LIDAR,λ)
Imitative Models (*Ours*)†	Imitative Plan+PID	Traj. Density Est.	Goal Likelihoods	A* Waypointer	(LIDAR,λ)

Summary

- We proposed “Imitative Models” to combine the benefits of IL and MBRL. Imitative Models are probabilistic predictive models able to plan interpretable expert-like trajectories to achieve new goals.
- Inference with an Imitative Model resembles trajectory optimization in MBRL, enabling it to both incorporate new goals and plan to them at test-time, which IL cannot.
- Learning an Imitative Model resembles offline IL, enabling it to circumvent the difficult reward-engineering and costly online data collection necessities of MBRL.
- ICLR 2020 Talk:
  - https://iclr.cc/virtual_2020/poster_Skl4mRNYDr.html