harder imitation learning

Recall: imitation learning

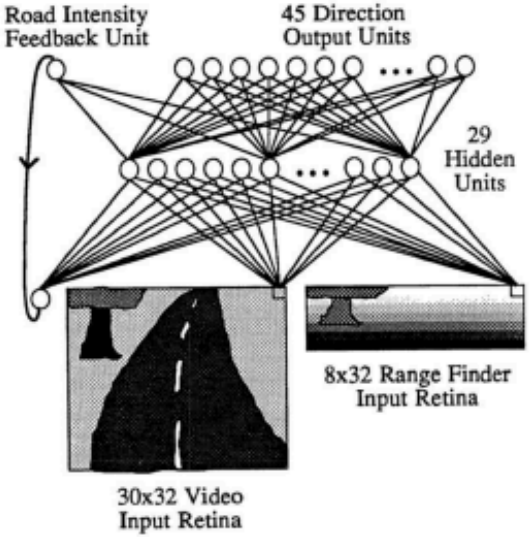
- Observe states, expert actions (s, a)
- Train classifier to predict a(s) using this data

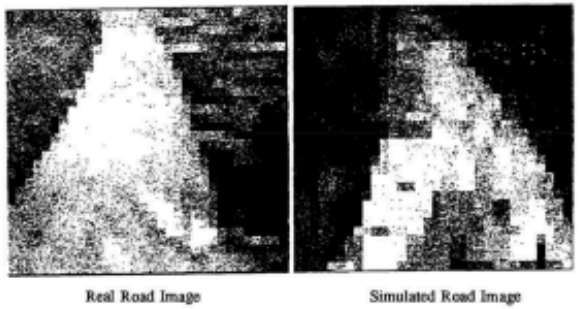
- Issues:
  - moving off-policy slightly leads to states that you haven't seen
  - so a(s) becomes unreliable
    - and so generates even more unfamiliar states -> catastrophe

- Fixes
  - multiple observations
  - Dagger

Demonstration Augmentation: ALVINN 1989

Road follower

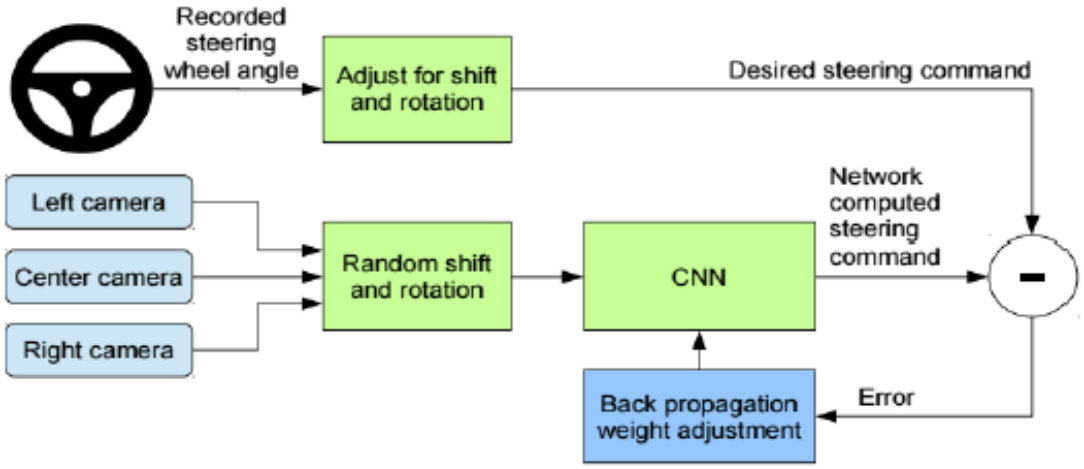


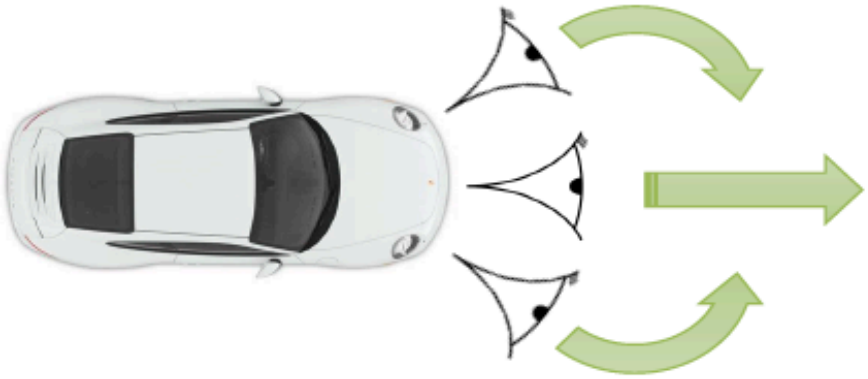


- Using graphics simulator for road images and corresponding steering angle ground-truth
- Online adaptation to human driver steering angle control
- 3 layers, fully connected layers, very low resolution input from camera and lidar..

"In addition, the network must not solely be shown examples of accurate driving, but also how to recover (i.e. return to the road center) once a mistake has been made. Partial initial training on a variety of simulated road images should help eliminate these difficulties and facilitate better performance." ALVINN: An autonomous Land vehicle in a neural Network, Pomerleau 1989

Demonstration Augmentation: NVIDIA 2016





Additional, left and right cameras with automatic grant-truth labels to recover from mistakes

"*DAVE-2 was inspired by the pioneering work of Pomerleau [6] who in 1989 built the Autonomous Land Vehicle in a Neural Network (ALVINN) system*. Training with data from only the human driver is not sufficient. The network must learn how to recover from mistakes. …",

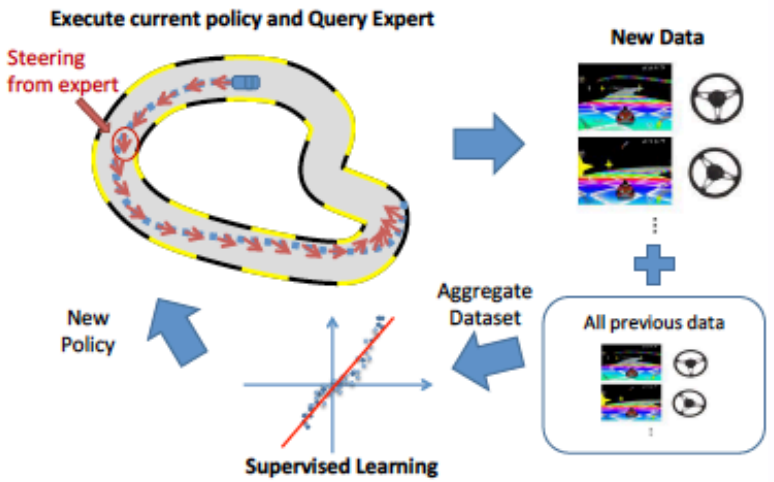
End to End Learning for Self-Driving Cars , Bojarski et al. 2016

Fragkiadaki, ND

DAGGER (in simulation)

Dataset AGGregation: bring learner's and expert's trajectory distributions closer by labelling additional data points resulting from applying the current policy

1. train $\pi_\theta(u_t|o_t)$ from human data $\mathcal{D}_{\pi^*} = \{o_1, u_1, ..., o_N, u_N\}$
2. run $\pi_\theta(u_t|o_t)$ to get dataset $\mathcal{D}_\pi = \{o_1, ..., o_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions u_t
4. Aggregate: $\mathcal{D}_{\pi^*} \leftarrow \mathcal{D}_{\pi^*} \cup \mathcal{D}_\pi$
5. GOTO step 1.



Problems:

- execute an unsafe/partially trained policy
- repeatedly query the expert

A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, Ross et al. 2011

There remain serious problems

- Expert's intent
- Bias in on-policy data
  - Dagger will collect too much lane following, not enough intersections

Crucial issue: intent

Codevilla, 18

Why has imitation learning not scaled up to fully autonomous urban driving? One limitation is in the assumption that the optimal action can be inferred from the perceptual input alone. This assumption often does not hold in practice: for instance, when a car approaches an intersection, the camera input is not sufficient to predict whether the car should turn left, right, or go straight. Mathematically, the mapping from the image to the control command is no longer a function. Fitting a function approximator is thus bound to run into difficulties. This had already been observed in the work of Pomerleau: "Currently upon reaching a fork, the network may output two widely discrepant travel directions, one for each choice. The result is often an oscillation in the dictated travel direction" [27]. Even if the network can resolve the ambiguity in favor of some course of action, it may not be the one desired by the passenger, who lacks a communication channel for controlling the network itself.

Codevilla, 18

Incorporating intent

Traditional story:

Obtain:

$$\mathcal{D} = \{(\mathbf{o}_i, a_i)\}$$

Compute:

$$\hat{\theta} = \text{argmin}_\theta \sum_i l(F(\mathbf{o}_i; \theta), a_i)$$

Controller is:

$$F(\mathbf{o}; \hat{\theta})$$

With intent: $\mathbf{c}_i$

Obtain:

$$\mathcal{D} = \{(\mathbf{o}_i, \mathbf{c}_i, a_i)\}$$

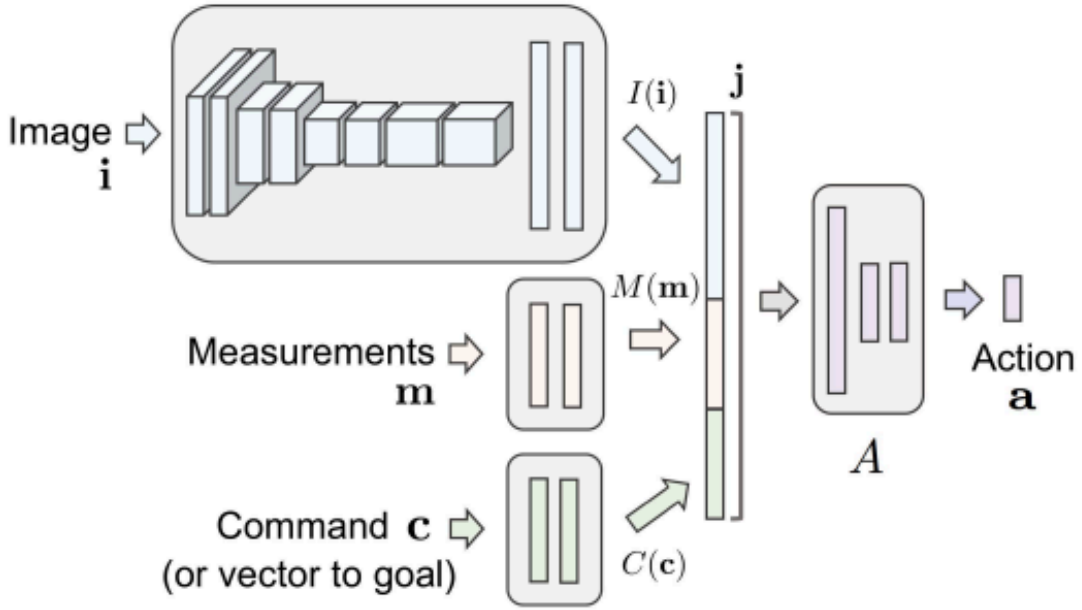
Compute:

$$\hat{\theta} = \text{argmin}_\theta \sum_i l(F(\mathbf{o}_i, \mathbf{c}_i; \theta), a_i)$$

Controller is:

$$F(\mathbf{o}, \mathbf{c}; \hat{\theta})$$

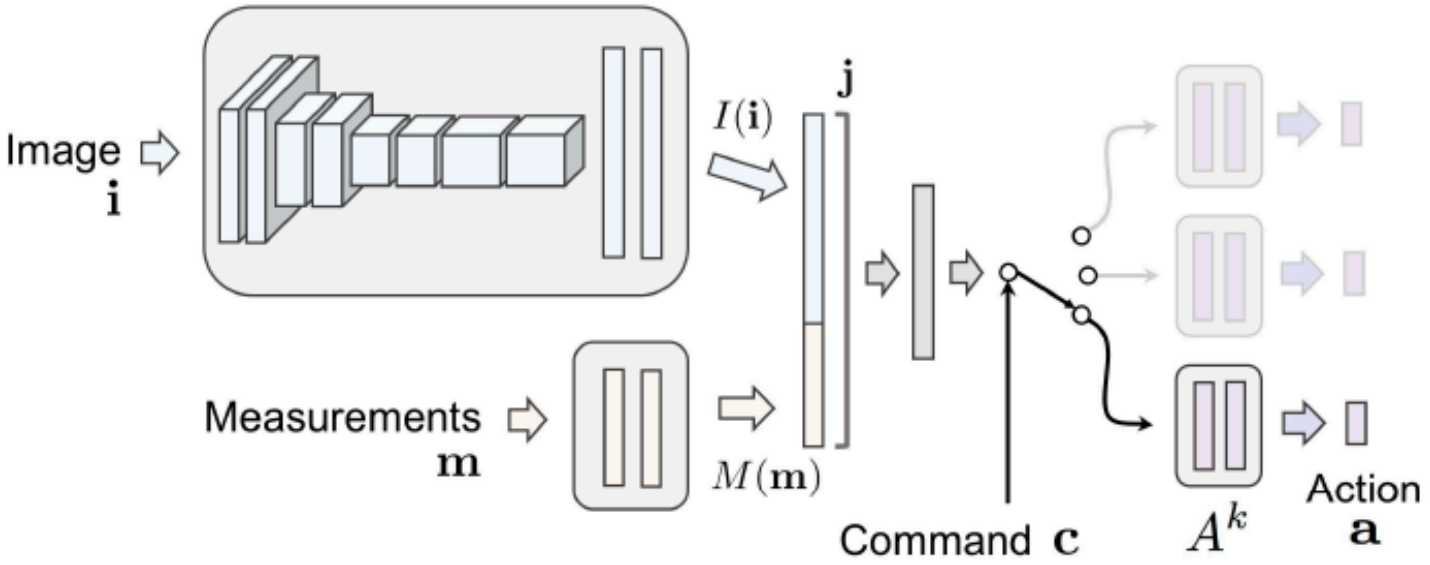
Intent via command input



(a)

Codevilla, 18

Branched intent



(b)

Codevilla, 18

Training

- Control
  - steering and acceleration
    - (likely a PID accepts this, turns into signals!)
- 3 Camera Trick
  - But this isn't good enough
- Aggressive data augmentation on images
- Noise injection during expert driving
  - essentially,
    - perturb vehicle response to expert
    - let expert recover

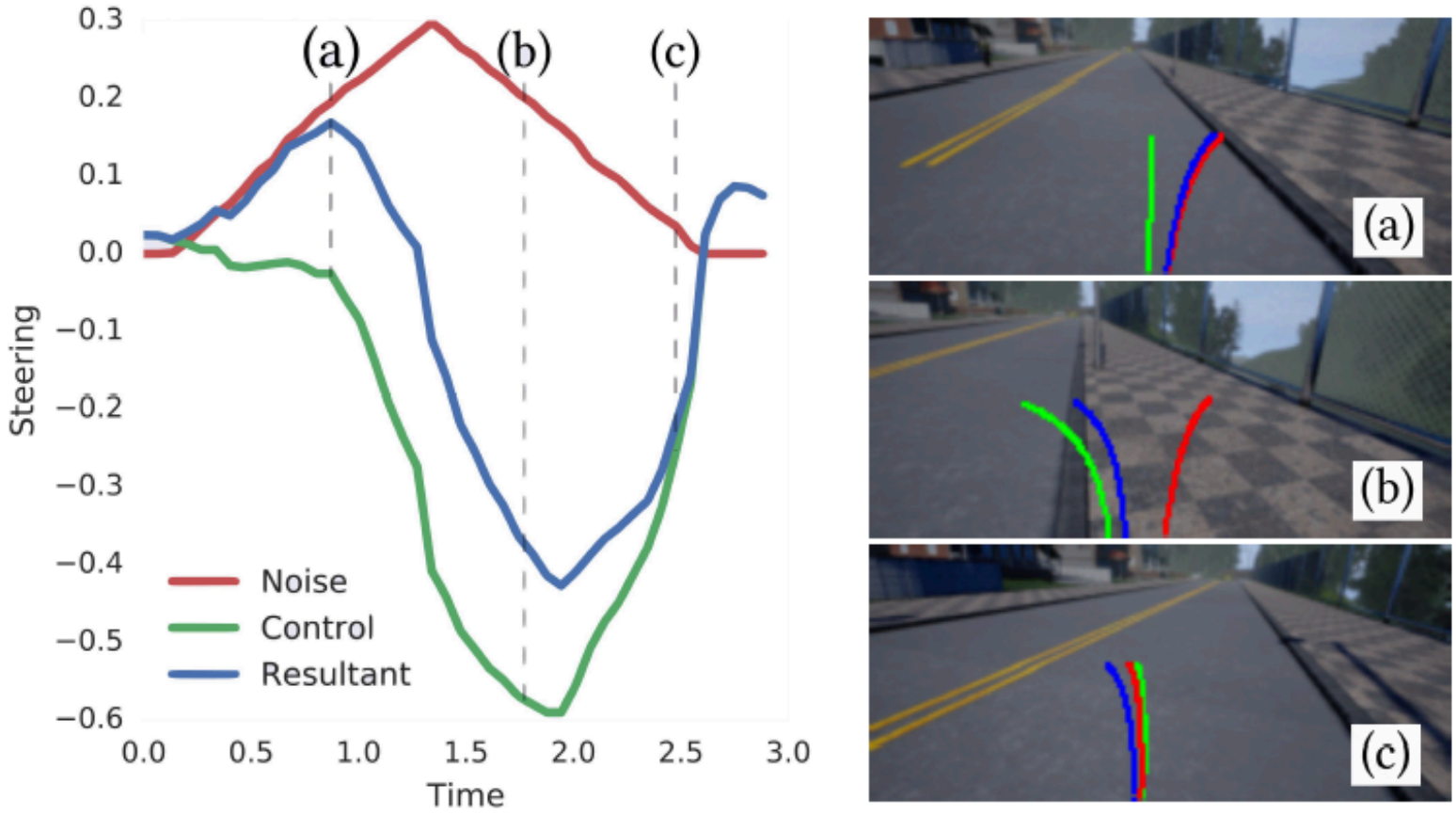


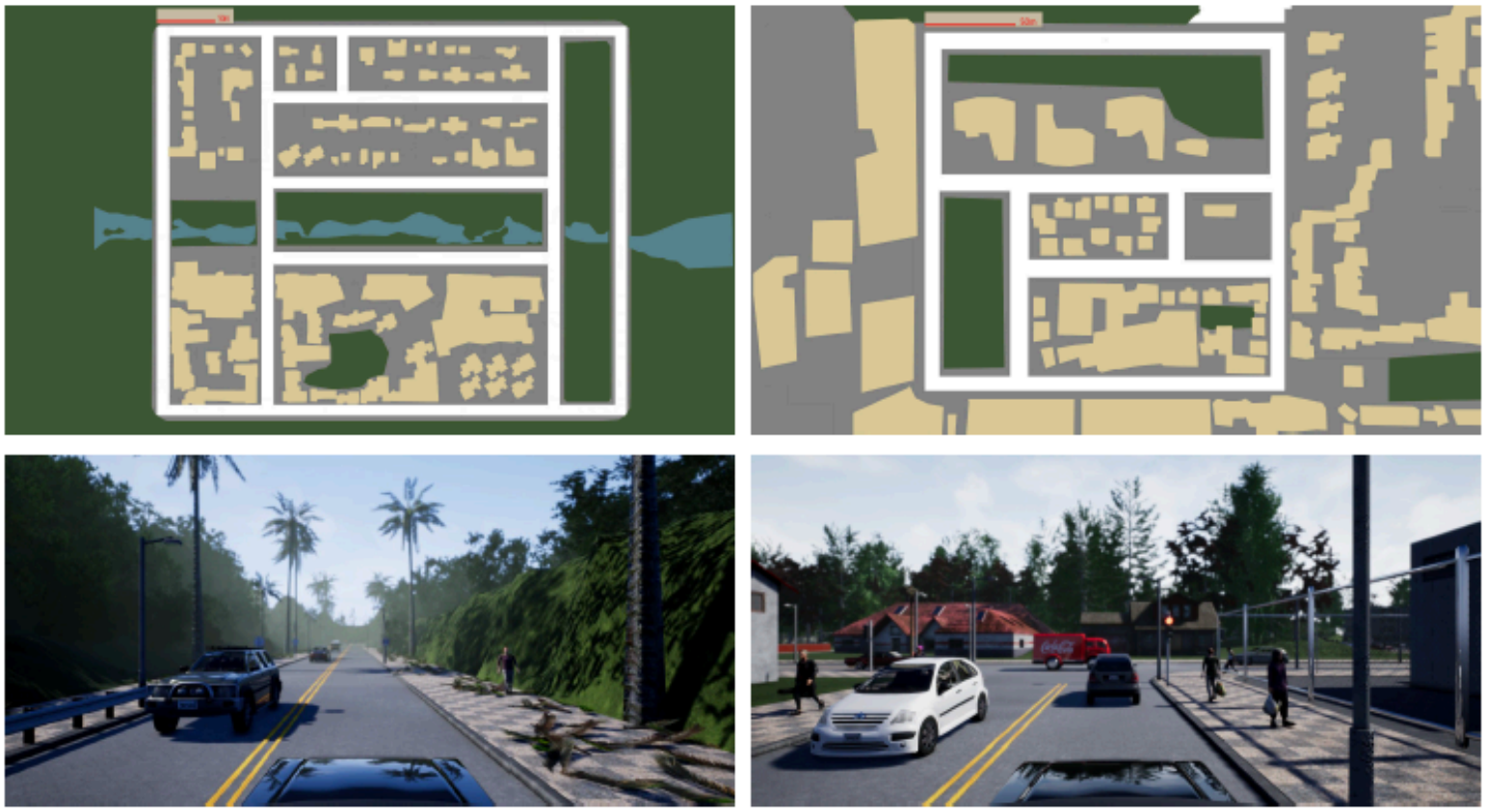
Fig. 4. Noise injection during data collection. We show a fragment from an actual driving sequence from the training set. The plot on the left shows steering control [rad] versus time [s]. In the plot, the red curve is an injected triangular noise signal, the green curve is the driver's steering signal, 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.

Codevilla, 18

CARLA

A. Simulated Environment

We use CARLA [10], an urban driving simulator, to corroborate design decisions and evaluate the proposed approach in a dynamic urban environment with traffic. CARLA is an open-source simulator implemented using Unreal Engine 4. It contains two professionally designed towns with buildings, vegetation, and traffic signs, as well as vehicular and pedestrian traffic. Figure 5 provides maps and sample views of Town 1, used for training, and Town 2, used exclusively for testing.



Town 1 (training) Town 2 (testing)

Fig. 5. Simulated urban environments. Town 1 is used for training (left), Town 2 is used exclusively for testing (right). Map on top, view from onboard camera below. Note the difference in visual style.

In order to collect training data, a human driver is presented with a first-person view of the environment (center camera) at a resolution of 800×600 pixels. The driver controls the simulated vehicle using a physical steering wheel and pedals, and provides command input using buttons on the steering wheel. The driver keeps the car at a speed below 60 km/h and strives to avoid collisions with cars and pedestrians, but ignores traffic lights and stop signs. We record images from the three simulated cameras, along with other measurements such as speed and the position of the car. The images are cropped to remove part of the sky. CARLA also provides extra information such as distance travelled, collisions, and the occurrence of infractions such as drift onto the opposite lane or the sidewalk. This information is used in evaluating different controllers.

Codevilla, 18

In addition to the observations (images) and actions (control signals), we record commands provided by the driver. We use a set of four commands: `continue` (follow the road), `left` (turn left at the next intersection), `straight` (go straight at the next intersection), and `right` (turn right at the next intersection). In practice, we represent these as one-hot vectors.

CARLA results

Model	Success rate		Km per infraction	
	Town 1	Town 2	Town 1	Town 2
Non-conditional	20%	26%	5.76	0.89
Goal-conditional	24%	30%	1.87	1.22
Ours branched	**88%**	**64%**	2.34	1.18
Ours cmd. input	78%	52%	3.97	1.30
Ours no noise	56%	22%	1.31	0.54
Ours no aug.	80%	0%	4.03	0.36
Ours shallow net	46%	14%	0.96	0.42

Table 1. Results in the simulated urban environment. We compare the presented method to baseline approaches and perform an ablation study. We measure the percentage of successful episodes and the average distance (in km) driven between infractions. Higher is better in both cases, but we rank methods based on success. The proposed `branched` architecture outperforms the baselines and the ablated versions.

Codevilla, 18

Not just simulation…

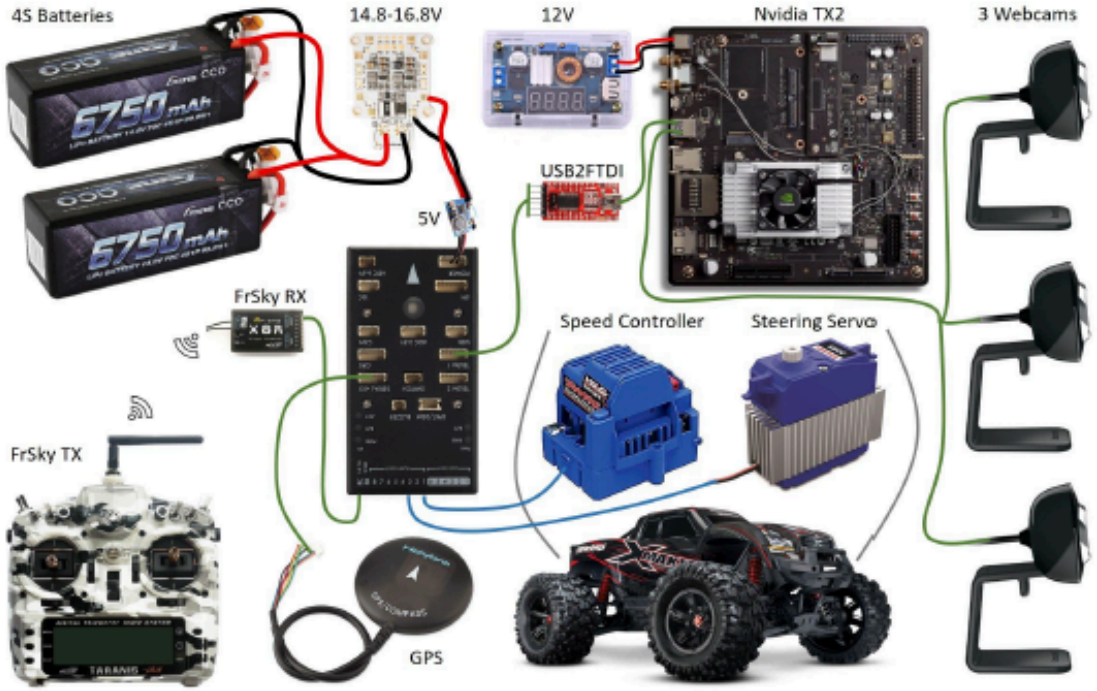


Fig. 6. Physical system setup. Red/black indicate +/- power wires, green indicates serial data connections, and blue indicates PWM control signals.

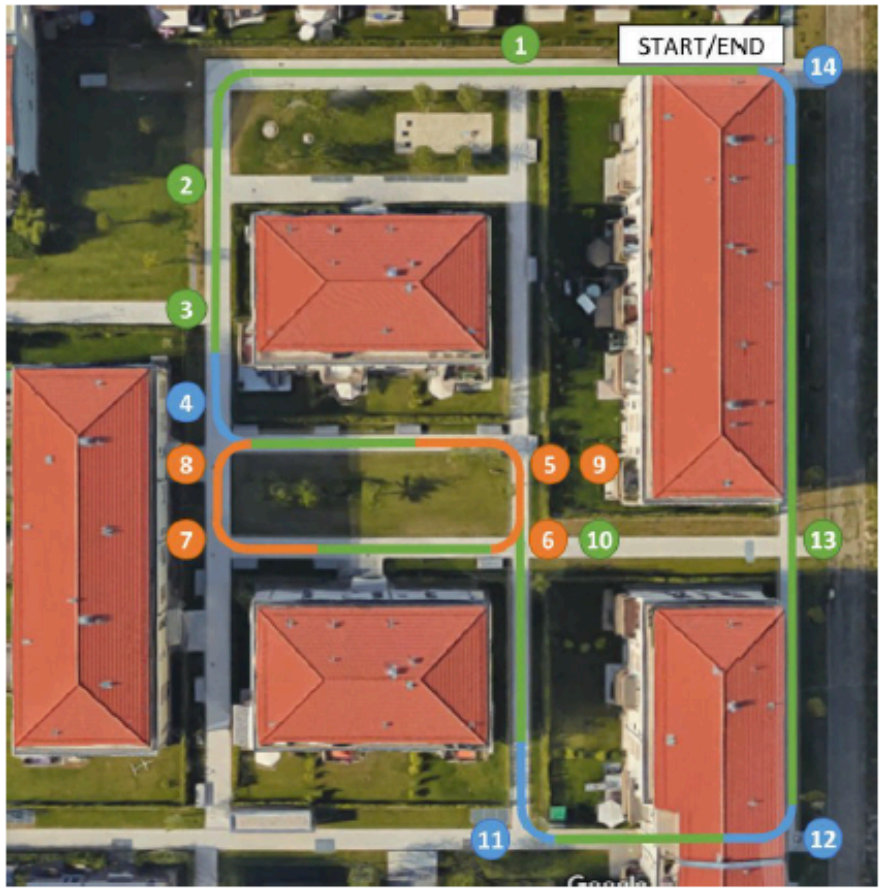
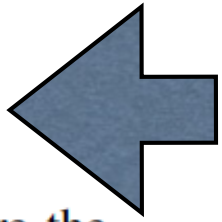


Fig. 8. A map of the primary route used for testing the physical system. Intersections traversed by the truck are numbered according to their order along the route. Colors indicate commands provided to the vehicle when it approaches the intersection: blue = `left`, green = `straight`, orange = `right`.

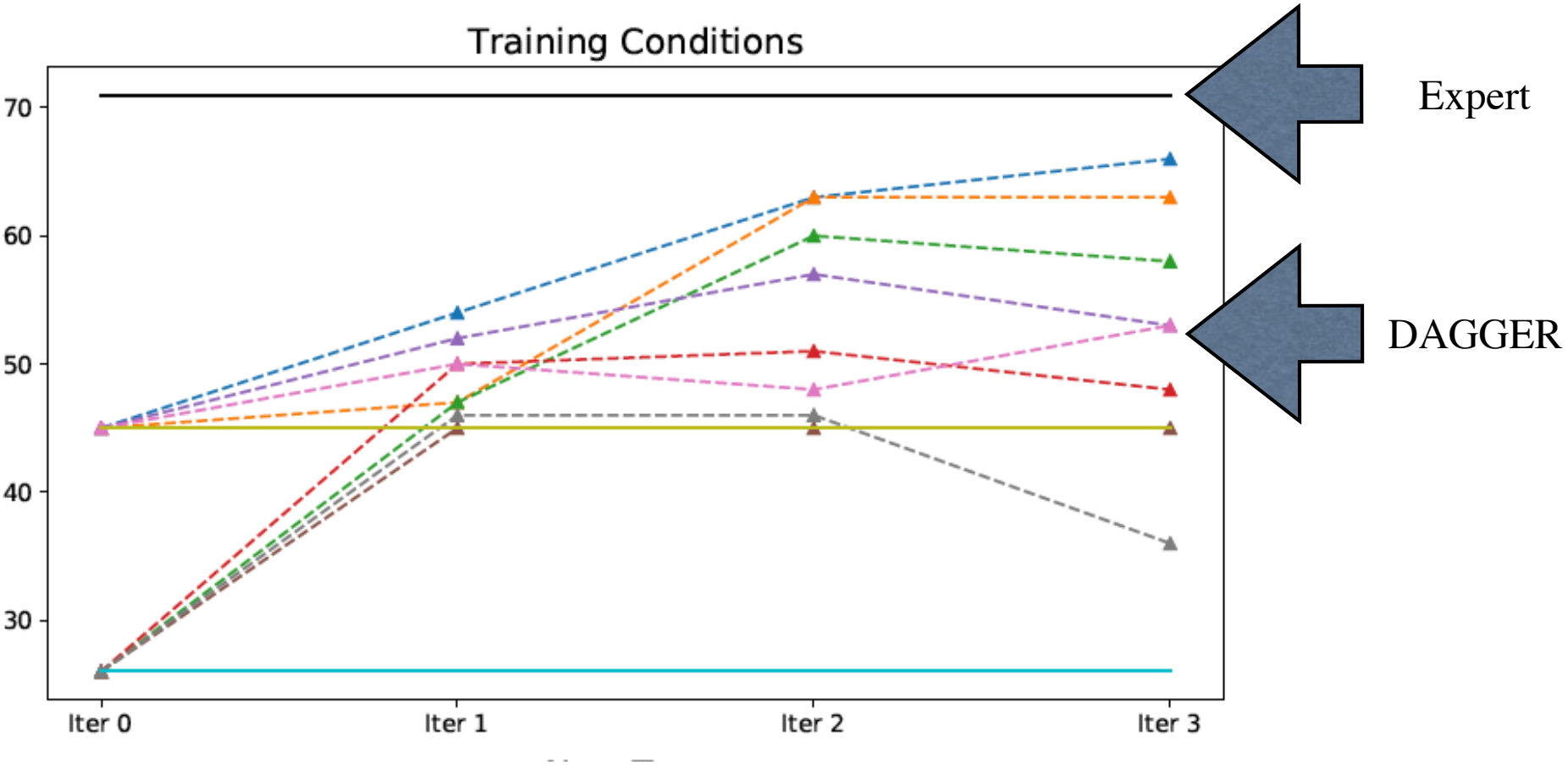
Model	Missed turns	Interventions	Time
Ours branched	**0%**	**0.67**	**2:19**
Ours cmd. input	11.1%	2.33	4:13
Ours no noise	24.4%	8.67	4:39
Ours no aug.	73%	39	10:41



Link to weather ideas

Table 2. Results on the physical system. Lower is better. We compare the `branched` model to the simpler `command input` architecture and to ablated versions (without noise injection and without data augmentation). Average performance across 3 runs is reported for all models except for "Ours no aug.", for which we only performed 1 run to avoid breaking the truck.

Straightforward DAGGER isn't that good



DAGGER Issue

in the training conditions. This happens because as DAgger continues to append on-policy data, the diversity of the dataset does not grow fast enough compared to the growth of the main mode of demonstrations, e.g., driving straight in lane. Consequently, the performance decreases as more data is collected since the driving policy is not able to learn how to react in rare modes, e.g., close proximity to dynamic agents. This result is in direct contrast to prior applications of DAgger in robotics [5, 18, 42, 46, 55] and reflects the limitation of DAgger in case of datasets having significant bias. This observation is also consistent with [12] where the authors show that additional data does not necessarily lead to improvement in performance for urban autonomous driving. Further, we observe that the performance of DAgger in the generalization conditions starts to drop after the second iteration. This is expected since the aggregated on-policy data is collected in the training conditions, thereby leading to overfitting as the dataset size increases.

Strategies

- Identify states where current policy is uncertain
  - “Critical states”
  - entropy of policy classifier
    - one might use other measures
- Subsample the collection of new states to fixed size
  - “replay buffer”

Algorithm 1 DAgger with Critical States and Replay Buffer

Collect D_0 using expert policy π^*

$\hat{\pi}_0 = \text{argmin}_{\pi}\ \mathcal{L}(\pi, \pi^*, D_0)$

Initialize replay buffer $D \leftarrow D_0$

Let $m = |D_0|$

for $i = 1$ *to* N **do**

  Generate on-policy trajectories using $\hat{\pi}_{i-1}$

  Get dataset $D_i = \{(s, \pi^*(s))\}$ of visited states by $\hat{\pi}_{i-1}$ and actions given by expert

  Get $D'_i \leftarrow \{(s_c, \pi^*(s_c))\}$ after sampling critical states from D_i

  Combine datasets: $D \leftarrow D \cup D'_i$

  while $|D| > m$ **do**

    Sample $(s, \pi^*(s))$ randomly from $D \cap D_0$

    $D \leftarrow D - \{(s, \pi^*(s))\}$

  end

  Train $\hat{\pi}_i = \text{argmin}_{\pi}\ \mathcal{L}(\pi, \pi^*, D)$ with policy initialized from $\hat{\pi}_{i-1}$

end

return $\hat{\pi}_N$

Concentrate on states where policy doesn't know what to do

Knock out states which are "duplicated" or "redundant"

Variants

- DA-CS
  - use critical states, but not replay buffer
- DA-RB
  - use both
- DA-RB+
  - use noise perturbations on expert as in Codevilla 18
- DA-RB+(E)
  - use ensemble of multiple DA-RB+ models (from training iterations)

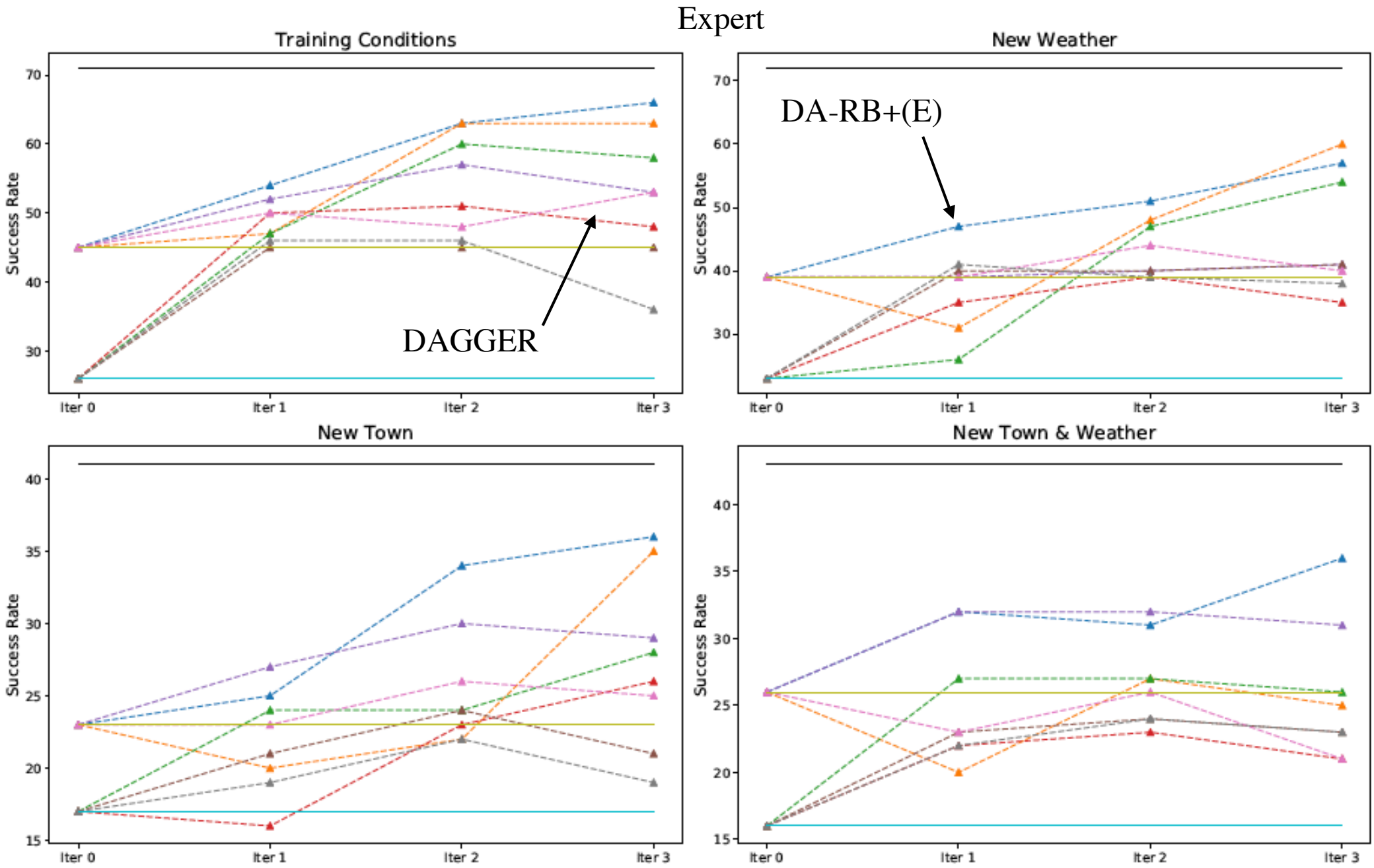


Figure 2: **Success rate of different methods across conditions.** '+' represents training with perturbed expert data.

Task	CILRS$^+$	DART	DA-RB$^+$ (Ours)	DA-RB$^+$(E) (Ours)	Expert
Train	45 ± 6	50 ± 1	62 ± 1	**66** ± 5	71 ± 4
NW	39 ± 4	37 ± 2	**60** ± 1	56 ± 1	72 ± 3
NT	23 ± 1	26 ± 2	34 ± 2	**36** ± 3	41 ± 2
NTW	26 ± 2	21 ± 1	25 ± 1	**35** ± 2	43 ± 2

Table 3: **Success rate on dense setting of all conditions.** Mean and standard deviation over 3 evaluation runs. NW-New Weather, NT-New Town, NTW-New Town & Weather, DA-RB$^+$(E) - ensemble of DA-RB$^+$ over all iterations.

Notice the massive impact of weather

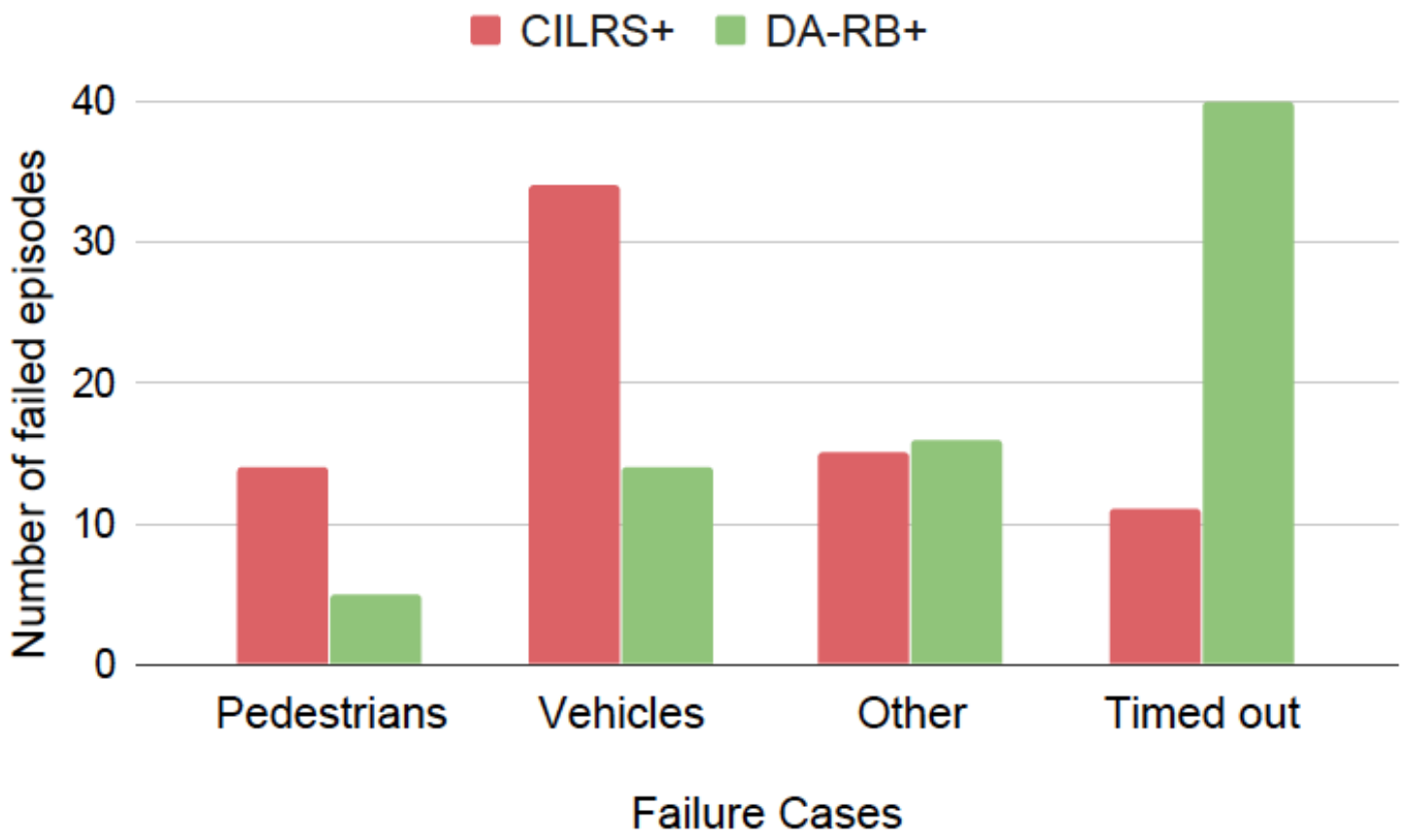
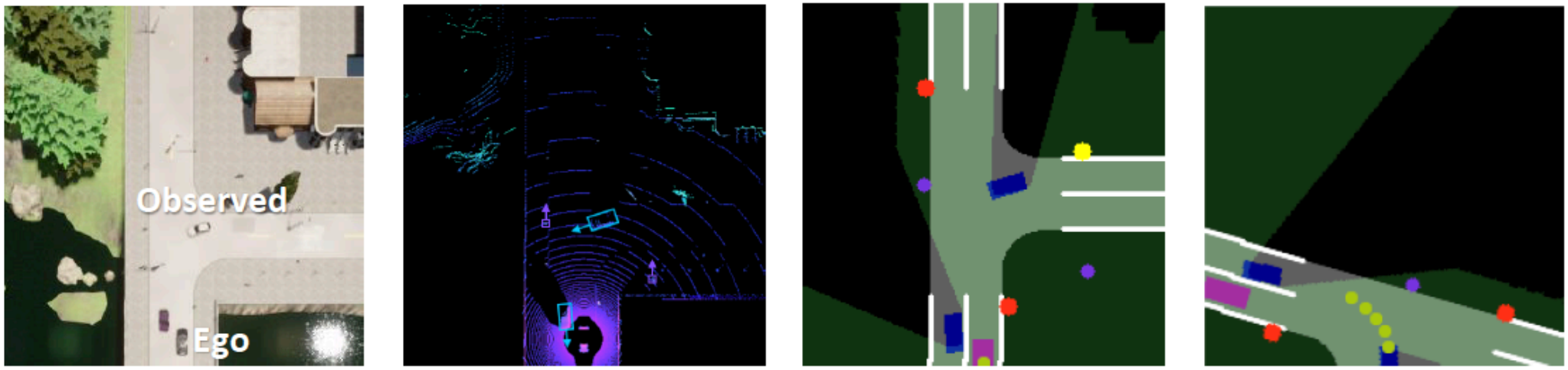


Figure 3: **Failure case analysis.** We consider collision with pedestrians, vehicles, other static objects and timed out scenarios on the dense setting of New Town & Weather.

Learning by watching

- Q: why use only expert data for imitation learning?
  - Other cars are also driven by experts - benefit from them
- Issues:
  - recover state representation for other cars
  - recover control intentions for other cars



(a) Reference scene (b) LiDAR map (c) Ego-centric BEV (d) Estimated BEV

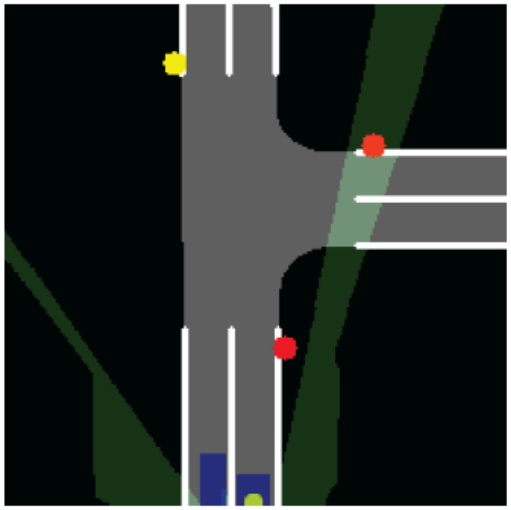
Figure 2: **Bird's-Eye-View (BEV) Representation With Visibility Map Overlay.** We visualize (a) the reference top-down view of a scene, (b) its corresponding LiDAR map with vehicle and pedestrian detections, (c) the ego-centric BEV after integration with map information, including the ego-vehicle (magenta), surrounding vehicles (blue), pedestrians (purple circles), traffic lights (yellow, red), and visibility map (green overlay), and (d) the estimated BEV from the perspective of the observed vehicle. Dark yellow circles show future waypoints (Section 3.3) for the ego and observed vehicle.

- Key property of a good BEV
  - you can figure out what the BEV from *another* vehicle looks like
  - from that, can figure out what the LIDAR, image, etc looks like
- Intention
  - represent as a sequence of waypoints
  - which you can get for another car by just waiting to see what it does
    - and tracking

Zhang 21

(a) Reference scene

(b) Ego-centric BEV



(c) Estimated BEV

(d) Refurbished waypoints

Figure 3: **Refurbishing Difficult Samples.** The example estimated view for an observed vehicle, shown in (c), is ambiguous due to occlusion. The proposed refurbishment process (Section 3.3) can correct the waypoint targets (d) to align with the estimated state and encourage proper agent behavior, i.e., to move forward and maintain a closer distance to the red light.

Occlusion creates issues. Observed vehicle state misses the occluded vehicle, so waypoints imputed by tracking are weird (you can't tell it's braking because something is in front of it, because you can't tell there is something in front of it). Waypoints could be fixed by smoothing from original non LbW model (waypoint refurbishment)

Q: does this create collision problems?
A: apparently not
Q: why?
A: < please supply >

Big strength: Data efficiency

Table 1: **Ablation Study.** Comparison of driving success rate (%) for the proposed approach (LbW) with various visibility fusion schemes. The baseline model (Ego) is trained by traditional behavior cloning. Mean and standard deviation are shown over three runs using the original CARLA benchmark (OB) and the NoCrash benchmark (Regular: NC-R, Dense: NC-D).

	One Hour			**30 Minutes**			**10 Minutes**		
	NC-R	NC-D	OB	NC-R	NC-D	OB	NC-R	NC-D	OB
Ego (Baseline)	46 ± 1	18 ± 1	56 ± 1	26 ± 2	12 ± 1	68 ± 1	24 ± 1	0 ± 0	64 ± 1
LbW	64 ± 1	24 ± 1	74 ± 1	52 ± 0	$\mathbf{24 \pm 0}$	68 ± 1	34 ± 1	6 ± 1	$\mathbf{82 \pm 4}$
LbW + Visibility (Early)	52 ± 1	$\mathbf{24 \pm 0}$	76 ± 1	54 ± 1	18 ± 1	72 ± 3	28 ± 1	6 ± 1	64 ± 1
LbW + Visibility (Late)	$\mathbf{92 \pm 3}$	$\mathbf{24 \pm 0}$	$\mathbf{92 \pm 1}$	$\mathbf{74 \pm 2}$	$\mathbf{24 \pm 0}$	$\mathbf{92 \pm 0}$	$\mathbf{52 \pm 1}$	$\mathbf{20 \pm 2}$	68 ± 1

Zhang 21

Waypoint refurbishment helps

Table 3: **Refurbishment Analysis.** Impact of waypoint refurbishment on driving success rate (%). Results are shown using the late fusion visibility integration scheme.

Town 2	NC-R	NC-D	OB
LbW + Visibility	64 ± 0	32 ± 3	86 ± 1
LbW + Visibility (Refurbishment)	$\mathbf{80 \pm 1}$	$\mathbf{36 \pm 0}$	$\mathbf{96 \pm 0}$
Town 3	**NC-R**	**NC-D**	**OB**
LbW + Visibility	40 ± 0	30 ± 1	$\mathbf{60 \pm 0}$
LbW + Visibility (Refurbishment)	$\mathbf{60 \pm 0}$	$\mathbf{40 \pm 0}$	$\mathbf{60 \pm 0}$

Zhang 21