End-to-End Model-Free Reinforcement Learning for Urban Driving using Implicit Affordances

Marin Toromanoff[1,2,3], Emilie Wirbel[2,3], Fabien Moutarde[1]
[1]Center for Robotics, MINES ParisTech, PSL [2]Valeo Driving Assistance Research [3]Valeo.ai
[1]name.surname@mines-paristech.fr [2,3]name.surname@valeo.com

Abstract

Reinforcement Learning (RL) aims at learning an optimal behavior policy from its own experiments and not rule-based control methods. However, there is no RL algorithm yet capable of handling a task as difficult as urban driving. We present a novel technique, coined implicit affordances, to effectively leverage RL for urban driving thus including lane keeping, pedestrians and vehicles avoidance, and traffic light detection. To our knowledge we are the first to present a successful RL agent handling such a complex task especially regarding the traffic light detection. Furthermore, we have demonstrated the effectiveness of our method by winning the Camera Only track of the CARLA challenge.

1. Introduction

Urban driving is probably one of the hardest situations to solve for autonomous cars, particularly regarding the interaction on intersections with traffic lights, pedestrians crossing and cars going on different possible lanes. Solving this task is still an open problem and it seems complicated to handle such difficult and highly variable situations with classic rules-based approach. This is why a significant part of the state of the art in autonomous driving [20, 4, 5] focuses on end-to-end systems, i.e. learning driving policy from data without relying on hand-crafted rules.

Imitation learning (IL) [28] aims to reproduce the behavior of an expert (a human driver for autonomous driving) by learning to mimic the control the human driver applied in the same situation. This leverages the massive amount of data annotated with human driving that most of automotive manufacturer and supplier can obtain relatively easily. On the other side, as the human driver is always in an almost perfect situation, IL algorithms suffer from a distribution mismatch, i.e. the algorithm will never encounter failing cases and thus will not react appropriately in those conditions. Techniques to augment the database with such failing cases do exist but they are currently mostly limited to lane keeping and lateral control [1, 34].

Deep Reinforcement Learning (DRL) on the other side lets the algorithm learn by itself by providing a reward signal at each action taken by the agent and thus does not suffer from distribution mismatch. This reward can be sparse and not describing exactly what the agent should have done but just how good the action taken is locally. The final goal of the agent is to maximize the sum of accumulated rewards and thus the agent needs to think about sequence of actions rather than instantaneous ones. One of the major drawbacks of DRL is that it can need a magnitude larger amount of data than supervised learning to converge, which can lead to difficulties when training large networks with many parameters. Moreover many RL algorithms rely on a replay buffer [21, 25, 12] allowing to learn from past experiments but such buffers can limit the size of the input used (e.g. the size of the image). That is why neural networks and image size in DRL are usually tiny compared to the ones used in supervised learning. Thus they may not be expressive enough to solve such complicated tasks as urban driving. Therefore current DRL approaches to autonomous driving are applied to simpler cases, e.g. only steering control for lane keeping [18] or going as fast as possible in racing games [24, 16]. Another drawback of DRL, shared with IL, is that the algorithm appears as a black box from which it is difficult to understand how the decision was taken.

A promising way to solve both the data efficiency (particularly for DRL) and the black box problem is to use privileged information as auxiliary losses also coined affordances in some recent papers [2, 31]. The idea is to train a network to predict high level information such as semantic segmentation maps, distance to center of the lane, traffic light state etc... This prediction can then be used in several ways, either by a classic controller as in Sauer et al. [31], either as auxiliary loss helping to find better features to the main imitative task loss as in Mehta et al. [23] or also in a model-based RL approach as in the really recent work of Pan et al. [26] while also providing some interpretable feedback on how the decision was taken.

In this work, we will present our RL approach for the case of end-to-end urban driving from vision, including lane keeping, traffic light detection, pedestrian and vehicle avoidance, and handling intersection with incoming traffic. To achieve this we introduce a new technique that we coin *implicit affordances*. The idea is to split the training in two phases: first an encoder backbone (Resnet-18 [11]) is trained to predict affordances such as traffic light state or distance to center of the lane. Then the output features of this encoder is used as the RL state instead of the raw images. Therefore the RL signal is only used to train the last part of the network. Moreover the features are used directly in the replay memory rather than the raw images, which corresponds to approximately 20 times less memory needed. We showed our method performance by winning the "Camera Only" track in the CARLA Autonomous Driving Challenge [30]. To our knowledge we are the first to show a successful RL agent on urban driving, particularly with traffic lights handling.

We summarize our main contributions below:

- The first RL agent successfully driving from vision in urban environment including intersection management and traffic lights detection.
- Introducing a new technique coined *implicit affordances* allowing training of replay memory based RL with much larger network and input size than most of network used in previous RL works.
- Extensive parameters and ablation studies of implicit affordances and reward shaping.
- Showcase of the capability or our method by winning the "Camera Only" track in the CARLA Autonomous Driving Challenge.

2. Related Work

2.1. End-to-End Autonomous Driving with RL

As RL relies on trial and error, most of RL works applied to autonomous cars are conducted in simulation both for safety reasons and data efficiency. One of the most used simulator is TORCS [36] as it is an open-source and simple to use racing game. Researchers used it to test their new actor-critic algorithm to control a car with discrete actions in Mnih et al. [24] and with continuous actions in Lillicrap et al. [21]. But as TORCS is a racing game, the goal of those works is to reach the end of the track as fast as possible and thus does not handle intersections nor traffic lights.

Recently, many papers used the new CARLA [7] simulator as an open-source urban simulation including pedestrians, intersections and traffic lights. In the original CARLA paper [7], the researchers released a driving benchmark along with one Imitation learning and one RL baseline. The RL baseline was using the A3C algorithm with discrete actions [24] and its results were far behind the imitation baseline. Lang et al [20] used RL with DDPG [21] and continuous actions to fine-tune an imitation agent. But they rely mostly on imitation learning and do not explicitly explain how much improvement comes from the RL fine-tuning. Moreover they also do not handle traffic lights.

Finally, there are still only few RL methods applied in a real car. The first one was Learning to Drive in a Day [18] in which an agent is trained directly on the real car for steering. A really recent work [37] also integrates RL on a real car and compares different ways of transferring knowledge learned in CARLA in the real world. Even if their studies are really interesting, their results are preliminary and applied only on few specific real-world scenarios. Both of these works only handle steering angle for lane keeping and a large gap has to be crossed before reaching throttle and steering control simultaneously in urban environment on a real car with RL.

2.2. Auxiliary Tasks and Learning Affordances

The UNREAL agent [15] is one of the first articles to study the impact of auxiliary tasks for DRL. They showed that adding losses such as predicting incoming reward could improve data efficiency and final performance on both Atari games and labyrinth exploration.

Chen et al. [2] introduce affordance prediction for autonomous driving: a neural network is trained to predict high level information such as distance to the right, center and left part of the lane or distance to the preceding car. Then they used those affordances as input to a rule-based controller and reached good performance on the racing simulator TORCS. Sauer et al. upgraded this in their Conditionnal Affordance Learning [31] paper to handle more complicated scenarios such as urban driving. In order to achieve that they also predict information specific to urban driving such as the maximum allowed speed and the incoming traffic light state. As Chen et al. they finally used those information in a rule-based controller and showed their performance in the CARLA benchmark [7] for urban driving. Both of those works do not include any RL and rely on rule-based controller. Just after, Mehta et al. [23] used affordances as auxiliary tasks to their imitation learning agent and showed it was improving both data efficiency and final performance. But they do not handle traffic lights and rely purely on imitation.

Finally, there are two really recent articles closely related to ours. The first one by Gordon et al [10] introduced SplitNet on which they explicitly decompose the learning scheme in finding features from perception task and use these features as input to their model-free RL agent. But their scheme is applied to a completely different task, robot navigation and scene exploration. The second one by Pan

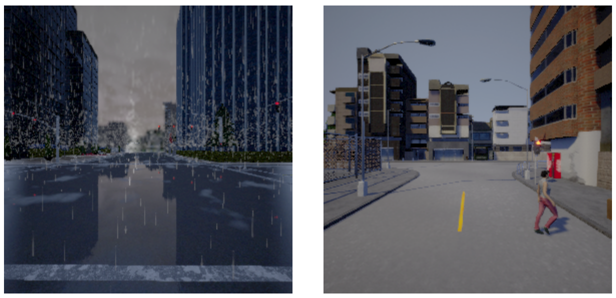

Figure 1. Sample of traffic light image (left is US, right is EU).

et al. [26] train a network to predict high-level information such as probability of collision or being off-road in the near futures from a sequence of observations and actions. They use this network in a model-based RL scheme by evaluating different trajectories to finally apply the generated trajectory giving the lowest cost. However, they use a model-based approach and do not handle traffic light signal.

3. The CARLA Challenge

The CARLA Challenge [30] is an open competition for autonomous driving relying on the CARLA simulator. This competition addresses specifically the problem of urban driving. The goal is to drive in unseen maps from sensors to control, ensuring lane keeping, handling intersections with high level navigation orders (Right, Left, Straight), handling lane changes, pedestrians and other vehicles avoidance and finally handling traffic lights US and EU at the same time (traffic lights are positioned differently in Europe and in US, see Figure 1). This is much more challenging than the original CARLA benchmark [7]. The CARLA Challenge consists in 4 different tracks with the only difference being the sensors available, from cameras only to a full stack perception. We will only handle the "Camera Only" track there, in fact we even used only a single frontal camera for all this work.

4. Method

In this section we describe our general approach (RL setup, reward shaping and network architecture). In the next section, we describe what adaptations are needed to make this approach usable in an autonomous driving context.

4.1. RL Setup: Rainbow-IQN Ape-X

There are two main families of model-free RL: value-based and policy-based methods. We choose to use value-based RL as it is the current state-of-the-art on Atari [12] and is known to be more data efficient than policy-based method. However, it has the drawback of handling only discrete actions. Making a comparison between value-based RL and policy-based RL (or actor-critic RL which is a sort

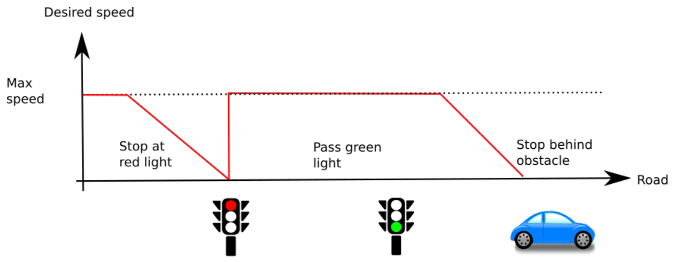

Figure 2. Desired speed according to environment. The desired speed adapts in function of the situation, getting lower when arriving close to a red light, going back to maximum speed when traffic light goes to green and again getting lower when arriving behind an obstacle. The speed reward is maximum when the vehicle speed is equal to the desired speed.

of combination of both) for Urban driving is out of the scope of this paper but would definitely be interesting for future work. We started with our open-source[1] implementation of Rainbow-IQN Ape-X [12, 6, 13] (for Atari originally) taken from our previous work [33]. We removed the dueling network [35] from Rainbow as we found it was leading to same performance while using much more parameters. The distributed version of Rainbow-IQN was mandatory for our usage: CARLA is too slow for RL and cannot generate enough data if only one instance is used. Moreover this allowed us to train on multiple maps of CARLA at the same time, generating more variability in the training data, better exploration and providing an easy way to handle both US and EU traffic lights (some town used in training were US while others were EU).

4.2. Reward Shaping

The reward used for the training relies mostly on the waypoint API present in the latest version of CARLA (CARLA 0.9.X). This API allows to get continuous waypoints position and orientation of all lanes in the current town. This is fundamental to decide what path the agent has to follow. Moreover, this API provides the different possibilities at each intersection. At the beginning of an episode, the agent is initialized on a random waypoint on the city, then the optimal trajectory the agent should follow can be computed using the waypoint API. When arriving at an intersection, we choose randomly a possible maneuvre (Left, Straight or Right) and the corresponding order is given to the agent. The reward relies on three main components: *desired speed*, *desired position* and *desired rotation*.

The *desired speed* reward is maximum (and equal to 1) when the agent is at the desired speed, and linearly goes down to 0 if the agent speed is lower or higher. The desired speed, illustrated on Figure 2, is adapting to the situation: when the agent arrives near a red traffic light, the desired speed goes linearly to 0 (the closest the agent is

[1]https://github.com/valeoai/rainbow-iqn-apex

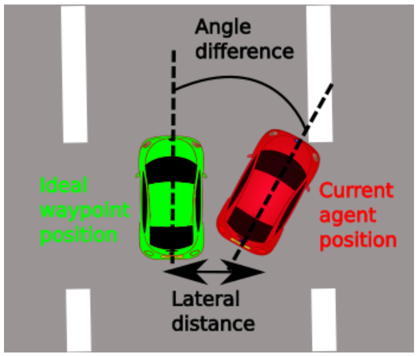

Figure 3. Lateral distance and angle difference for lateral and angle reward computation. The difference is measured between the ideal waypoint (in green) and the current agent position (in red).

from the traffic light), and goes back to maximum allowed speed when it turns green. The same principle is used when arriving behind an obstacle, pedestrian, bicycle or vehicle. The desired speed is set to a constant *maximum speed* (here 40km/h) on all other situations.

The second part of the reward, the *desired position*, is inversely proportional to the distance from the middle of the lane (we compute this distance using the waypoints mentioned above). This reward is maximum equal to 0 when agent is exactly in the middle of the lane and goes to -1 when reaching a maximum distance from lane D_{max}. When the agent is further than D_{max}, the episode terminates. For all our experiments, D_{max} was set to 2 meters: this is the distance from the middle of the lane to the border. Other termination conditions are colliding with anything, running a red light and being stuck for no reason (i.e. not behind an obstacle nor stopped at a red traffic light). For all those termination conditions, the agent receives a reward of -1.

With only the two previous reward components, we observed the trained agents were not going straight as oscillations near the center of lane were giving almost the same amount of reward as going straight. That is why we added our third reward component, *desired rotation*. This reward is inversely proportional to the difference in angle between the agent and the orientation of the nearest waypoint from the optimal trajectory (see Figure 3 for details). Ablation studies on the reward shaping can be found at section 6.3.

4.3. Network Architecture

Most of networks used in model-free RL with images as input train a particularly small network [7, 12] compared to networks used commonly in supervised learning [32, 11]. One of the larger networks used for model-free RL for Atari is the *large architecture* from IMPALA [8] which consists of 15 convolutional layers and 1.6 million parameter: as comparison our architecture has 18 convolutional layers and 30M parameters. Moreover IMPALA used more than 1B frames when we used only 20M. The most common architecture (e.g. [24, 6]) is the one introduced in the original DQN paper [25], taking a 84×84 grayscale image as input. Our first observation is that traffic light state (particularly for US traffic lights which are farther) can not be seen on so small images. Therefore a larger input size has been chosen (around 40 times larger): $4 \times 288 \times 288 \times 3$ by concatenating 4 consecutive frames as a simple and standard [25, 7] way to add some temporality in the input. We choose this size as it was the smallest one we tested on which we still had a good accuracy on traffic light detection (using a conventional supervised training). We choose to use Resnet-18 [11] as a relatively small network (compared to the one used in supervised training) to ensure a small inference time. Indeed RL needs a lot of data to converge so each step must be as fast as possible to reduce the overall training time. However, even if Resnet-18 is among the smallest networks used for supervised learning, it contains around 140 times more weights in its convolutional layers than DQN [25]. Moreover Resnet-18 incorporates most of state-of-the art advances in supervised learning such as residual connections and batchnorm [14]. Finally, we use a conditional network as in Codevilla et al. [4] to handle 6 different maneuvers: follow lane, left/right/straight, change lane left/right. The full network architecture is described in Figure 4.

5. Challenges and Solutions to apply RL to Complex Autonomous Driving Tasks

In this section, we present our suggestions to solve the issues arising when using a large network with RL and how to handle discrete actions.

5.1. Training RL with high complexity input size: Implicit Affordances

How to train a larger network with larger images for RL? Using a larger network and input size raises two major issues. The first one is that such a network is much longer and harder to train. Indeed it is well known that training a DRL agent is data consuming even with tiny networks. The second issue is the replay memory. One of the major advantages of value-based RL [25, 12] over policy-based methods is to be off-policy, meaning the data used for learning can come from another policy. However storing image 35 times bigger raises issues for storing as many transitions (usually 1M transitions are stored which correspond to 6GB for 84×84 images and thus would be 210GB for $288 \times 288 \times 3$ images which is unpractical).

Our main idea is to pre-train the convolutional encoder part of the network to predict some high-level information and then freeze it while training the RL. The intuition is that the RL signal is too weak to train the whole network but can be used to train only the fully connected part. Moreover this solves the replay memory issue as we can now store features directly in the replay memory and not the raw images.

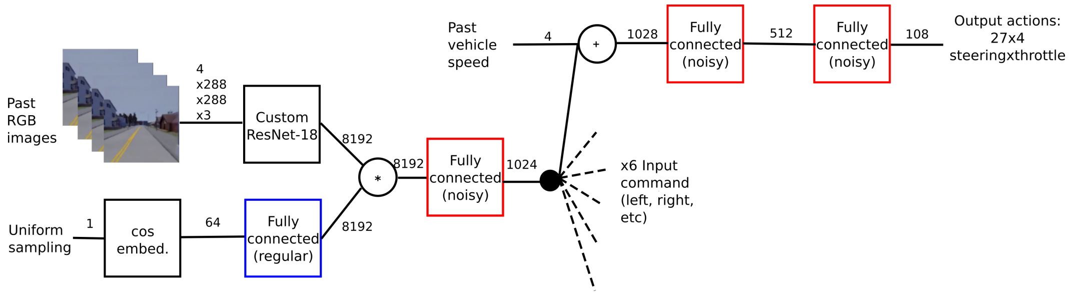

Figure 4. Network architecture. A Resnet-18 [11] encoder is used in a conditional network [4] with a Rainbow-IQN [33] RL training (hence the IQN network [6] and noisy fully connected layers [9])

We coin this scheme as *implicit affordances* because the RL agent does not use the explicit predictions but has only access to the implicit features (i.e the features from which our initial supervised network predicts the explicit affordances).

Which high level semantic information/affordances to predict? The most simple idea to pre-train our encoder would be to use an auto-encoder [19], i.e. trying to compress the images by trying to predict back the full image from a smaller feature space. This was used in the work Learning to Drive in a Day [18] and allowed for faster training on their real car. We argue this would not work for our harder use-case particularly regarding the traffic light detection. Indeed, traffic light states represent only a few pixels in the image (red or green) but those pixels are the most relevant for the driving behavior.

To ensure that there is relevant signal in the features used as RL state, we choose to rely on high level semantic information available in CARLA. We use 2 main losses for our supervised phase: traffic light state (binary classification) and semantic segmentation. Indeed all relevant information but traffic light state is contained in our semantic segmentation. We use 6 classes for the semantic mask: moving obstacles, traffic lights, road markers, road, sidewalk and background. We also predict some other affordances to help the supervised training such as the distance to the incoming traffic light, if we are in an intersection or not, the distance from the middle of the lane and the relative rotation to the road. The two last estimations are coming from our viewpoint augmentation (without it the autopilot is always perfectly in the middle of the lane with no rotation). Our supervised training with all our losses is represented in Figure 5. Ablation studies to estimate the impact of these affordance estimations are presented on section 6.2.

Viewpoints Augmentation The data for the supervised phase is collected while driving with an existing autopilot in the CARLA simulator. However this autopilot always stays in the middle of the lane, so the pre-trained encoder which is frozen does not generalize well during the RL training,

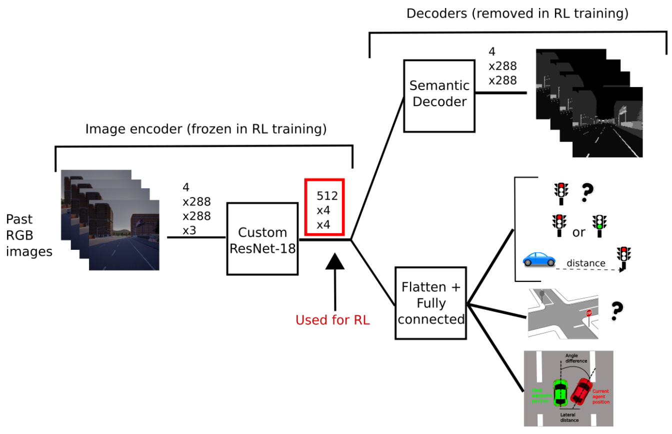

Figure 5. Decoder and losses used to train the encoder: semantic segmentation, traffic light (presence, state, distance), intersection presence, lane position (distance and rotation)

particularly when the agent starts to deviate from the middle of the lane: with an encoder trained on data collected only from autopilot driving, an RL agent performance is poor. This is the exact same idea as for IL with the distribution mismatch and the intuition behind it is explained on Figure 6. To solve this, we suggest to add viewpoints augmentation by moving the camera around the autopilot. With this augmentation the encoder performance is much better while the RL agent drives and explores and we found this was mandatory to obtain good performance during the RL training phase.

In summary, training a large encoder with well selected supervised tasks and using the resulting feature maps, the implicit affordances, as an input state for RL training, addresses the problem of input, network and replay memory size. One should take care to properly augment the data during the supervised training phase, to make sure the encoding is adapted during the exploration of the RL training.

5.2. Handling Discrete Actions

As aforementioned, standard value-based RL algorithms such as DQN [25], Rainbow [12] and Rainbow-IQN [33] imply to use discrete actions. Preliminary experiment with

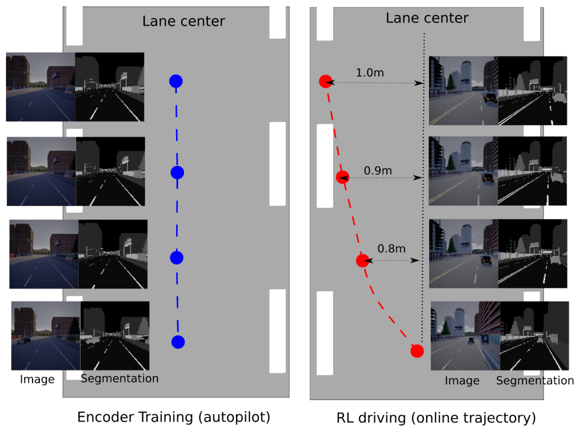

Figure 6. Why data augmentation is needed for training the encoder: RL agents trajectories (right) might deviate from the lane center, which leads to semantic segmentation with much more varied lane marking positions than what can be encountered if training only from autopilot data (left).

few discrete actions (only 5 for steering) resulted in agents oscillating and failing to stay in lane. Better results can be obtained by using more steering actions such as 9 or 27 different steering values. Throttle is less of an issue: 3 different values for throttle are used, plus one for brake. This leads to a total of 36 (9×4) or 108 (27×4) actions for our experiments. We also try to predict the derivative of steering angle: the prediction of network is used to update the previous steering (which is given as input) instead of using directly the prediction as current steering. The impact of these choices is studied in section 6.3.

To reach more fine-grained discrete actions, we strongly suggest to use a bagging of multiple predictions and average them. To do so, we can simply use consecutive snapshots of the same training, which avoids having to train again and is free to have. This trick is consistently improving behavior, reducing oscillations by a large margin and obtaining better final performance. Furthermore, as the encoder is frozen, it can be shared, so the computational overhead of averaging multiple snapshots of the same training is almost negligible (less than 10% of the total forward time for averaging 3 predictions). Therefore, all our reported results are obtained by averaging 3 consecutive snapshots of the same training together (for example, results at 10M steps is the bagging of snapshots at 8M, 9M and 10M).

In summary, discrete actions can be compensated by increasing the number of actions, and averaging several discrete predictions.

6. Experiments and Ablation Studies

6.1. Defining a Common Test Situation and a Metric for Comparison

We first define a common set of scenarios and a metric to make fair comparison. Indeed the CARLA challenge maps are not publicly available and the old CARLA benchmark is only available on a deprecated version of CARLA (0.8.X) on which rendering and physics differs from the version of CARLA used in the CARLA challenge (0.9.X). Moreover as aforementioned, this CARLA benchmark is a much simpler task than the CARLA challenge.

Defining test scenarios We choose the hardest environment in the available maps of CARLA. Town05 includes the biggest urban district, is mainly multi-lane and US style: the traffic lights are on the opposite side of the road and much harder to detect. We also randomly spawn pedestrians crossing the road ahead of our agent to verify our models brake on this situations. We additionally set changing weather to make the task as hard as possible. This way, even with a single town training, we have a challenging setup. The single town training is necessary to make all our experiments and ablations studies in a reasonable time. All those experiments were made with 20M iterations on CARLA, with 3 actors (so 6.6M steps for each actor) and with a framerate of 10 FPS. Thus 20M steps is equivalent to around 20 days of simulated driving (as a comparison the most standard time [25, 6] used to train RL for Atari games is 200M frames corresponding to around 40 days and can go to more than 5 years of gametime [17, 13]). We define 10 scenarios of urban situations each one consisting in 10 consecutive intersections over the whole Town05 environment. We also define some scenarios on highway but those cases are much easier and thus less discriminative: for example our best model goes off-road less than one time every 100km on highway situation. Highway scenarios are mostly used for evaluating the oscillations of our different agents.

Defining a metric to compare different model and ablation studies We test our models 10 times on each scenario varying the weather condition and resetting the position of all other agents. Contrary to the training phase, we only terminate episode when the agent goes off-road as this allows to keep track of the number of infractions encountered. Our main metric is the average percentage of intersections successfully crossed (*Inters.*, higher is better), for example 50% completion corresponds to a mean of 5 intersections crossed in each scenario. We also keep track of the percentage of traffic lights passed without infraction (*TL*, higher is better) and the percentage of pedestrians passed without collision (*Ped.*, higher is better). Note that the last two are slightly less relevant, as a non-moving car will never run

Encoder used	Inters.	TL	Ped.
Random	0%	NA	NA
No TL state	33.4%	80%	82%
No segmentation	41.6%	96.5%	63%
All affordances	61.9%	97.6%	76%

Table 1. Comparison of agent performance with regards to encoder training loss (random weights, trained without traffic light loss, without semantic segmentation loss, or with all affordance losses)

Input/output	Inters.	TL	Ped.	Osc.
Constant desired speed	50.3%	31%	42%	1.51°
No angle reward	64.7%	99%	77.7%	1.39°
27 steering values (derivative)	64.5%	98.7%	85.1%	1.64°
9 steering values (absolute)	74.4%	98.5%	84.6%	0.88°
27 steering values (absolute)	75.8%	98.3%	81.6%	0.84°

Table 2. Performance comparison according to the steering angle discretization used and reward shaping

a red traffic light nor crash a pedestrian. That is why *Inters.* is our main metric for comparison: *TL* and *Ped.* are used for more fine-grained comparison. We also introduce a measure for oscillations: the mean absolute rotation between the agent and the road along the episode (*Osc.*, lower is better).

6.2. Ablations Studies on the Supervised Phase

In this section, we will detail our ablation studies concerning the supervised learning phase of affordances. The RL setup is exactly the same to ensure fair comparison.

First, some experiments are conducted without any supervised phase, i.e. training the whole network from scratch in the RL phase. Three different architectures are compared: the initial network from DQN with 84×84 images, a simple upgrade of the DQN network which takes $288 \times 288 \times 3$ images as input and finally our model with the Resnet-18 encoder.

Figure 7 shows that without affordances learning, agents fail to learn and do not even succeed to pass one intersection in average (less than 10% intersections crossed). Moreover it is important to note that training the bigger image encoder (respectively the full resnet-18) took 50% (resp. 200%) more time than training with our implicit affordances scheme even considering the time used for the supervised phase. Consequently these experiments are stopped after 10M steps. These networks also require much more memory, because full images are stored in the replay memory. As expected, these experiments prove that training a large network using only RL signal is hard.

The second stage of experiments concerned the Resnet-18 encoder training. First, as a sanity check, the encoder is frozen to random features. Then, either the traffic light state or the segmentation is removed from the loss in the supervised phase. These experiments show the interest of predicting the traffic light state and the semantic segmentation in our supervised training. The performance of the corresponding agents is illustrated in Figure 7.

Table 1 shows that removing the traffic light state has a huge impact on the final performance. As expected the RL agent using an encoder trained without the traffic light loss is running more red traffic lights. It is interesting to note that this ratio is much better than a random choice (which would be 25% of success for traffic light because traffic lights are green only 25% of the time). This means that the agent still succeeds to detect some traffic light state signal in the features. We guess that as the semantic segmentation includes a traffic light class (but not the actual state of it) the features contain some information about traffic light state. Removing the semantic segmentation loss from the encoder training also has an impact on final performance. As expected, performance on pedestrian collision is worse than any other training meaning the network has trouble to detect pedestrians and vehicles (this information is only contained in the semantic map).

6.3. Ablations Studies on the RL Setup

For fair comparison, the same pre-trained encoder is used for all experiments, trained with all affordances mentioned in Section 5.1. The encoder used here is the same one as the CARLA challenge, and has been trained on slightly more data and for more epochs than the encoders used for the previous ablation study.

Two experiments are conducted with different rewards to measure the impact of the reward shaping. In the first one (*constant desired speed*), the desired speed is not adapted to the situation: the agent needs to understand only from termination signal to brake on red traffic lights and to avoid collisions. In the second experiment, the *angle reward* component is removed to see the impact of this reward on oscillations. Two different settings for actions are also evaluated. First, the derivative of the steering angle is predicted instead of the current steering. Finally the steering angle discretization is studied, decreasing from 27 to 9 steering absolute values. Results are summarized in Table 2.

The most interesting result of these experiments is the one from *Constant desired speed*. Indeed, the agent fails totally at braking for both cases of red traffic light or pedestrian crossing: its performance is much worse than any other agent. The agent trained with desired speed set to constant runs 70% of traffic lights which is very close to a random choice. It also collides with 60% of pedestrians. This experiment shows how important the speed reward component is to learn a braking behaviour.

Surprisingly, we find that predicting derivative of steering results in more oscillations, even more than when removing the *desired rotation* reward component. Finally, taking 9 or 27 different steering values does not have any significant impact and both of these agents reach the best performance with low oscillation.

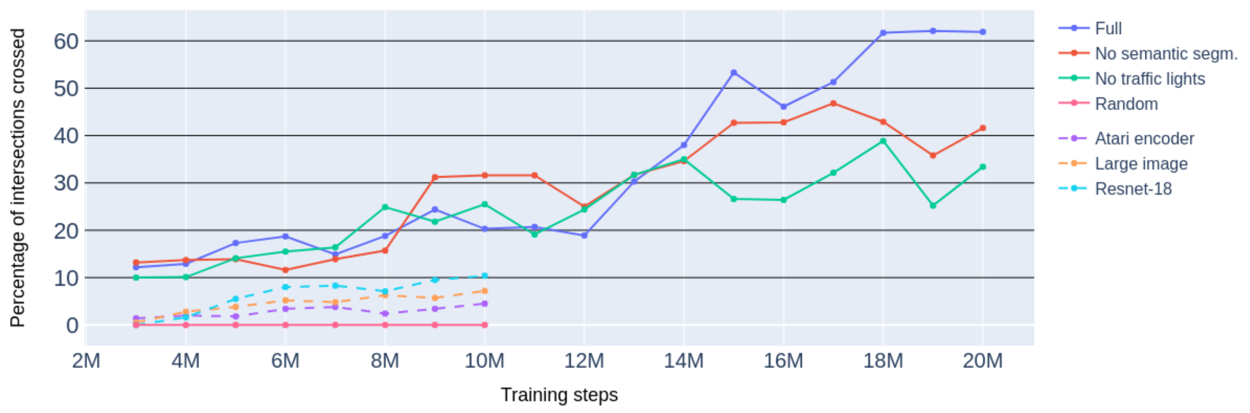

Figure 7. Evolution of agent performance with training steps and choice of the encoder behavior. The first group of encoders (solid lines) have frozen weights, the second group (dashed) are trained only by the RL signal (stopped earlier because the performance is clearly lower). Some experiments are averaged over multiple seeds (see Supplementary Materials for details on stability).

Training	Unseen EU Town	Unseen US Town
Only Town05	2.4%	42.6%
Multi town	58.4%	36.2%

Table 3. Generalization performance (*Inters.* metric).

6.4. Generalization on Unseen Towns

Finally, we conduct experiments on generalization, following the actual setting of the CARLA challenge. For this purpose, we train on 3 different towns at the same time (one with EU traffic light and the 2 others with US) and test on 2 unseen town (one EU and one US). We also test our best single town agent as a generalization baseline.

Results are presented in Table 3. We can see that performance on the unseen EU town is really poor for the agent trained only on a single US town, confirming the interest of training on both EU and US town at the same time. On the unseen US town, the performance is roughly similar for both trainings. These experiments show that our method generalizes to unseen environments.

6.5. Comparison on CARLA Benchmark

Very recently, Learning by Cheating (LBC) [3] re-implemented on open-source the CARLA benchmark on the newest version of CARLA (0.9.6). With such limited time, we did not have time to change our training setup at time of submission regarding the weather condition, so only *training weather* results are reported in Table 4 (test weather results can be found in the Supplementary).

LBC [3] which uses IL, is the only one outperforming our RL agent on the hardest task of *CoRL2017* benchmark (ie. *Nav. dynamic*). We also have similar results to the LBC baseline on the much harder *NoCrash* benchmark. Note that

	CoRL2017 (train town)					NoCrash (train town)		
Task	RL	CAL	CILRS	LBC	Ours	Task	LBC	Ours
Straight	89	100	96	100	100	Empty	97	100
One turn	34	97	92	100	100	Regular	93	96
Navigation	14	92	95	100	100	Dense	71	70
Nav. dynamic	7	83	92	100	100			
	CoRL2017 (test town)					NoCrash (test town)		
Task	RL	CAL	CILRS	LBC	Ours	Task	LBC	Ours
Straight	74	93	96	100	100	Empty	100	99
One turn	12	82	84	100	100	Regular	94	87
Navigation	3	70	69	98	100	Dense	51	42
Nav. dynamic	2	64	66	99	98			

Table 4. Success rate comparison (in % for each task and scenario, more is better) with baselines [7, 31, 5, 3] on train weathers.

we can only compare to LBC because other works have not been tested yet on the *NoCrash* benchmark with pedestrians (only available in the open-source implementation of LBC [3]). Finally, our work is outperforming the only other RL baseline [7] by a huge margin. This is also the first time a RL approach matches and even outperforms IL approaches on the CARLA benchmark. The inference code and the weights of our model can be found on open-source[2].

7. Conclusion

In this work, we present the first successful RL agent at end-to-end urban driving from vision including traffic light detection, using a value-based Rainbow-IQN-Apex training with an adapted reward and a large conditional network architecture. To solve this in a challenging autonomous driving context, we introduce *implicit affordances*, which use a large encoder trained for tasks relevant to autonomous driv-

[2]https://github.com/valeoai/LearningByCheating

ing in a supervised setting. We validate our design choices with ablation studies, and showcased our performance by winning the track “Camera Only” in the CARLA challenge.

In future work, it could be interesting to apply our *implicit affordances* scheme for policy-based or actor-critic and to train our affordance encoder on real images in order to apply this method on a real car.

Acknowledgements

The authors would like to thank Mustafa Shukor for his valuable time and his help on training some of our encoder.

References

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[3] Dian Chen, Brady Zhou, and Vladlen Koltun. Learning by Cheating. Technical report, 2019.

[4] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[5] Felipe Codevilla, Eder Santana, Antonio M. Lpez, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving, 2019.

[6] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. *arXiv preprint arXiv:1806.06923*, 2018.

[7] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[8] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018.

[9] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration, 2017.

[10] Daniel Gordon, Abhishek Kadian, Devi Parikh, Judy Hoffman, and Dhruv Batra. Splitnet: Sim2sim and task2task transfer for embodied visual navigation. *arXiv preprint arXiv:1905.07512*, 2019.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[12] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. 2018.

[13] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[15] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[16] Maximilian Jaritz, Raoul de Charette, Marin Toromanoff, Etienne Perot, and Fawzi Nashashibi. End-to-end race driving with deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2070–2075. IEEE, 2018.

[17] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning, 2019.

[18] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE, 2019.

[19] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

[20] Xiaodan Liang, Tairui Wang, Luona Yang, and Eric Xing. Cirl: Controllable imitative reinforcement learning for vision-based self-driving. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 584–599, 2018.

[21] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[22] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond, 2019.

[23] Ashish Mehta, Adithya Subramanian, and Anbumani Subramanian. Learning end-to-end autonomous driving using guided auxiliary supervision. *arXiv preprint arXiv:1808.10393*, 2018.

[24] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[26] Xinlei Pan, Xiangyu Chen, Qizhi Cai, John Canny, and Fisher Yu. Semantic predictive control for explainable and efficient policy learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3203–3209. IEEE, 2019.

[27] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation, 2016.

[28] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[29] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[30] German Ros, Vladfen Koltun, Felipe Codevilla, and Antonio Lopez. The CARLA Autonomous Driving Challenge. https://carlachallenge.org/, 2019.

[31] Axel Sauer, Nikolay Savinov, and Andreas Geiger. Conditional affordance learning for driving in urban environments. *arXiv preprint arXiv:1806.06498*, 2018.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[33] Marin Toromanoff and Emilie Wirbel. Is Deep Reinforcement Learning Really Superhuman on Atari? Leveling the playing field. Technical report, 2019.

[34] Marin Toromanoff, Emilie Wirbel, Frédéric Wilhelm, Camilo Vejarano, Xavier Perrotton, and Fabien Moutarde. End to end vehicle lateral control using a single fisheye camera. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3613–3619. IEEE, 2018.

[35] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning, 2015.

[36] Bernhard Wymann, Christos Dimitrakakis, Andrew Sumner, Eric Espié, and Christophe Guionneau. TORCS: The open racing car simulator. 2015.

[37] Bła Zej, Osinski Osinski, Adam Jakubowski, Christopher Galias, Silviu Homoceanu, and Henryk Michalewski. Simulation-based reinforcement learning for real-world autonomous driving. Technical report, 2019.

A. Supplementary materials: Implementation details

In this section, we will detail the hyper-parameters and the architecture of both the Supervised and the Reinforcement Learning training.

A.1. Supervised phase of affordances training: architecture and hyper-parameters

Our encoder architecture is mainly based on Resnet-18 [11] with two main differences. First, we changed the first convolutional layer to take 12 channels as input (we stack 4 RGB frames). Secondly, we changed the kernel size of downsample convolutional layers from 1x1 to 2x2. Indeed as mentionned in the paper Enet [27], *When downsampling, the first 1x1 projection of the convolutional branch is performed with a stride of 2 in both dimensions, which effectively discards 75% of the input. Increasing the filter size to 2x2 allows to take the full input into consideration, and thus improves the information flow and accuracy.*. We also removed the two last layers: the average pooling layer and the last fully connected. Finally, we added a last downsample layer taking 512x7x7 feature maps as input and outputting our RL state of size 512x4x4.

For the loss computation, we add a weight of 10 for the part of the loss around traffic light state detection, and 1 for all other losses.

Table 5. Supervised training hyperparameters

Parameter	Value
Learning rate	5.10^{-5}, eps 3.10^{-4} (Adam)
Batchsize	32
Epochs	20

For the semantic decoder, each layer consists of an upsample layer with a nearest neighbor interpolation, then 2 convolutional layers with batchnorm. All the other losses are build with fully connected layers with one hidden layer of size 1024. See Table 5 for more details on other hyper-parameters used in the supervised phase.

To train our encoder, we used a dataset of around 1M frames with associated ground-truth label (e.g. semantic segmentation, traffic light state and distance). This dataset was collected mainly in 2 cities of the CARLA [7] simulator: Town05 (US) and Town02 (EU).

A.2. Reinforcement Learning phase: architecture and hyper-parameters

In all our RL trainings, we used our encoder trained on affordances learning as a frozen image encoder: the actual RL state is the 8162 features coming from this frozen encoder. We then give this state to one fully connected layer of size 8162x1024. Then from these 1024 features concatenated with the 4 previous speed and steering angle values, we use a gated network to handle different orders as presented in CIL [4]. All the 6 heads have the same architecture but different weights, they are all made with 2 fully connected layers with one hidden layer of size 512.

Table 6. RL training hyperparameters for our *Single Town* and *Multi-Town* experiments: all parameters not mentioned come from the open-source implementation of Rainbow-IQN [33].

Parameter	Single Town / Multi-Town
Learning rate	5.10^{-5}, eps 3.10^{-4} (Radam)
Batchsize	32
Memory capacity	90 000 / 450 000
Number actors	3 / 9
Number steps	20M (23 days) / 50M (57 days)
Synchro. actors/learner	Yes / No

All hyperparameters used in our Rainbow-IQN training are the same as the one used in the open-source implementation [33] but for the replay memory size and for the optimiser. We use the really recent Radam [22] optimiser as it is giving consistent improvement on standard supervised training. Some comparisons were made with the Adam optimiser but did not show any significant difference. For all our *Single Town* experiments, we used Town05 (US) as environment. For our *Multi-Town* training, we used Town02 (EU), Town04 (US) and Town05 (US). Table 6 details the hyper-parameters used in our RL training.

B. Experiments

B.1. Stability study

One RL training of 20M steps was taking more than one week on a Nvidia 1080 Ti. That is why we did not have time nor computational resources to run an extensive study on the stability for all our experiments. Moreover evaluating our saved snapshot was also taking time, around 2 days to evaluate performance each million of steps as in Figure 7 of the main paper. Still, we performed multiple runs for 3 experiments presented in Table 1: *No TL state*, *No segmentation* and *All Affordances*. We evaluated those seeds at 10M and at 20M steps and the results (mean and standard deviation) can be found in the following Table 7.

	10M steps		20M steps	
Encoder used	Inters.	Nb seeds	Inters.	Nb seeds
No TL state	17.9% $\pm$ 7.3	6	27% $\pm$ 5.7	5
No segmentation	27.7% $\pm$ 9.3	5	41.7% $\pm$ 0.1	2
All affordances	24.9% $\pm$ 8.2	6	64.4% $\pm$ 2.5	2

Table 7. Mean and standard deviation of agents performance with regards to encoder training loss (trained without traffic light loss, without semantic segmentation loss, or with all affordance losses)

Even if we just have few different runs, those experiments on stability support the fact that our training are roughly stable and our results are significant. At 20M steps the "best" seed of *No TL state* perform worse than both seeds of *No segmentation*. More importantly, both seeds of *No segmentation* perform way worse than both seeds of *All affordances*.

B.2. Additional experiments

We made one experiment, *4 input one output*, to know the impact of predicting only one semantic segmentation instead of predicting 4 at the same time. Indeed, we stack 4 frames as our input and we thought it would give more information to learn from, if we train using all 4 semantic segmentations. We also tried to remove temporality in the input: taking only one frame as input and thus predicting only one semantic segmentation, *One input one output*. Finally, we made an experiment, *U-net Skip connection*, on which we used a standard U-net like architecture [29] for the semantic prediction. Indeed we did not use skip connections in all our experiments to prevent the semantic information to flow in this skip connections. Our intuition was that the semantic information could not be present in our final RL state (the last features maps of 4x4) if using skip connections.

The results of this 3 experiments are described in Table 8.

Encoder used	Inters.	TL	Ped.
One input one output	29.6%	95%	85%
4 input one output	64.3%	93.8%	70.7%
U-net Skip connection	58.6%	95%	69.8%
All affordances	64.4%	98.1%	76.2%

Table 8. Additional experiments to study impact of temporality both as input and as output of our Supervised phase. Also experiments with skip connection for the semantic prediction (U-net like skip connection [29]).

We can see from this results that using only one frame as input has a large impact on the final performance (going from 64% intersections crossed with our standard scheme *All Affordances* to 29% when using only one image as input). The impact of predicting only one semantic segmentation instead of 4 is marginal on our main metric (*Inters.*) but we can see that the performance on traffic lights (*TL*) and on pedestrians (*Ped.*) are slightly lower. Finally, the impact of using U-net like skip connections seems to be relatively small on the number of intersection crossed. However, there is still a difference with our normal system particularly on the pedestrians metric.

As a conclusion, those additional experiments confirmed our intuitions first about adding temporality both as input and output of our encoder and secondly to not use standard U-net skip connection is our semantic segmentation decoder to prevent semantic information to flow away from our final RL state. However, the impact of those intuitions are relatively small and we conducted only one seed which could not be representative enough.

B.3. Description of our test scenario

Each of our scenario is defined by a starting waypoint and 10 orders one for each intersection to cross. An example of one of our 10 scenario can be found on Figure 8. We also spawn 50 vehicles in the whole Town05 while testing. Finally, we spawn randomly pedestrian ahead of the agent every 20/30 seconds.

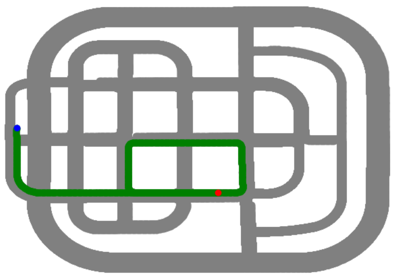

Figure 8. Sample of one of our scenario in Town05. The blue point is the starting point, the red is the destination.

B.4. Comparison on CARLA Benchmark: Implementation Details and Test Weathers Results

B.4.1 Test weathers results (train and test town)

As mentioned in the main paper, we did not have time to re-implement our training setup for the really recently released [3] implementation of the CARLA benchmark on the newer version of CARLA (0.9.6), particularly regarding the weather condition. At submission time, all our training were done under all possible weather conditions. That's why we reported our results only for training weathers in the main paper. We only had time to train our whole pipeline in the exact condition of the CARLA benchmark (i.e. only Town01 and train weathers for training and Town02 and test weathers for test) after acceptance. That's why we give our results for test weather only in the Supplementary Materials.

We can see from Table 9 that we are the only approach reaching a perfect score on all the tasks under test weathers. However, we can see that our results on the *NoCrash* benchmark fall far behind LBC [3] baseline under test weathers (even if our results were similar under train weathers). We found that the test weathers on the *NoCrash* benchmark are actually really different from the train weathers, particularly regarding sun reflection on the ground. We discovered that our frozen encoder trained only on Town01/train weathers was predicting sun reflection as "moving obstacles" and thus in this situation the RL agent is just braking for ever, acting like if a car was ahead. Most of our failure

	CoRL2017 (train town)					NoCrash (train town)		
Task	RL	CAL	CILRS	LBC	Ours	Task	LBC	Ours
Straight	86	100	96	100	100	Empty	87	36
One turn	16	96	96	96	100	Regular	87	34
Navigation	2	90	96	100	100	Dense	63	26
Nav. dynamic	2	82	96	96	100			

	CoRL2017 (test town)					NoCrash (test town)		
Task	RL	CAL	CILRS	LBC	Ours	Task	LBC	Ours
Straight	68	94	96	100	100	Empty	70	24
One turn	20	72	92	100	100	Regular	62	34
Navigation	6	88	92	100	100	Dense	39	18
Nav. dynamic	4	64	90	100	100			

Table 9. Success rate comparison (in % for each task and scenario, more is better) with baselines [7, 31, 5, 3] on test weathers.

under test weathers on *NoCrash* benchmark are in fact time-out because our agent is not moving anymore when he faces sun reflection on the ground. Handling diverse weather conditions is a known issue for perception algorithms and we think that improving our supervised performance (particularly the semantic segmentation) would probably manage this issue but this is left as future work.

B.4.2 Implementation Details for the CARLA benchmark

To train our new encoder in the exact condition of the CARLA benchmark, we used a new dataset of around 500K frames with associated ground-truth label (e.g. semantic segmentation, traffic light state and distance). This dataset was collected only in Town01 and under training weathers. Then we trained our RL agent with the *implicit affordances* coming from this new encoder for around 40M steps using 9 actors with all actors on Town01 under training weathers. We used a slightly bigger field of view (from 90° to 100°) and we cropped the sky (from 288x288x3 images to 288x168x3) as the EU traffic lights are less high than the US traffic lights (the CARLA benchmark contains only EU traffic lights). Finally, we removed all the change lane orders because all towns in CARLA benchmark are single lane (the CARLA benchmark setup is actually simpler than the CARLA challenge for which this paper has been initially done).

B.5. Training infrastructure

The training of the agents was split over several computers and GPUs, containing in total:

- 3 Nvidia Titan X and 1 Nvidia Titan V (training computer)
- 1 Nvidia 1080 Ti (local workstation)
- 2 Nvidia 1080 (local workstations)
- 3 Nvidia 2080 (training computer)