Reinforcement Learning for Traffic Signal Control

moving closer to responsible real-world deployment

October 30, 2023

vorgelegt von:	Sebastian Jost
geboren am:	04.09.0000 in Berlin
Matrikel-Nr.:	5019112
Studiengang:	Computational Modeling and Simulation, Track: Computational Mathematics
Anschrift:	Hainstr. 47, 12439 Berlin
E-Mail:	sebastian.jost@mailbox.tu-dresden.de
Verantwortlicher Hochschullehrer:	Prof. Dr. Ostap Okhrin
Betreuer:	Ankit Anil Chaudhari
Eingreicht:	Berlin, den ....

1 Abstract

Contents

List of Figures

List of Tables

2 Introduction and motivation

Moving people from one point to another is extremely important in our current society. Doing so efficiently can lower frustration in drivers, get people and vehicles to their destination faster and by that sometimes even save lives (e.g. fire trucks getting to their destination quicker). Using existing roads more efficiently can also affect city planning, as it encourages building new roads narrower than what would be necessary with inefficient traffic management. To ensure safety and allow vehicles from all direction to pass at intersections, traffic lights are a common choice - many cities worldwide have many hundreds or thousands of signaled intersections. However, traffic lights require vehicles to stop, increasing emisions and travel times. Recent advances in applying Reinforcement Learning to Traffic Signal Control [3], [16], [2] have shown, that there is a lot of room for improvement to increase efficiency and decrease overall travel time and even emissions caused by vehicles [12] by applying machine learning to traffic signal control. While some of the proposed Reinforcement Learning techniques have already been tested successfully in the real world, there are still many real-world complications that, to the best of our knowledge, have not yet been addressed in these methods. What we investigate here, are sensor failures and noisy sensor outputs. As shown in [13], all tested traffic sensors have a non-zero error rate and just like all other real-world devices, they also have a chance to completely fail [6], [7]. Additionally, we investigate the expected power consumption of running these learning based controllers in comparison to other electronic devices installed at intersections.

2.1 Definitions

In this section, we provide a few basic definitions of terms that are commonly used throughout the report. We define the true positive rate and false positive rate the same as in [13, pg. 39, 65], so we can use their data in our experiments.

True positive rate (detection rate)

$$p_{\text{TPR}} = \frac{D}{V} = \frac{\text{detected vehicles}}{\text{actual vehicles}}, \tag{1}$$

where D is the number of vehicles that were detected over a time period t and V is the number of vehicles that actually passed through the network in the same period. In other words, the true positive rate (TPR) is the proportion of actual vehicles that were detected.

False positive rate (misdetection rate)

$$p_{\text{FPR}} = \frac{W}{D} = \frac{\text{incorrectly detected vehicles}}{\text{detected vehicles}}, \tag{2}$$

where D is, as before, the detected number of vehicles in a time period and W is the number of vehicle detections, where no vehicle was actually present. In other words, the false positive rate (FPR) is the proportion of misdetections of all detections.

Pressure of an Intersection:

For an intersection i, let A_i be the set of all incoming lanes and B_i the set of all outgoing lanes at that intersection. Then, we define the pressure of the intersection as

$$P_i = \sum_{l_{in} \in A_i} x(l_{in}) - \sum_{l_{in} \in B_i} x(l_{out}), \tag{3}$$

where $x(l_{in})$ is the number of vehicles on each incoming lane l_{in}, $x(l_{out})$ is the number of vehicles on each outgoing lane l_{out} at intersection i.
Note that the definition of pressure in [17, Eq. 2] incorrectly includes the absolute value. This absolute value does not appear in the available code for *PressLight* [17], the *LibSignal* [10] library or in definitions elsewhere [19].

3 Related work

Applying reinforcement learning to traffic signal control has seen a rise in popularity in the past few years [14], [8], [2] with techniques incrementally improving and moving closer to real world applications on the scale of entire cities [3].
Many different state spaces, action spaces and reward functions have been explored along with different agent architectures. Some methods like [17], [14] and [3] try to incorporate domain knowledge of traditional traffic signal control techniques like [15]. Other techniques focus on improving coordination [16] between agents or scalability [3]. All of these techniques assume perfect knowledge of the environment. This is easy in a simulated environment, regardless of whether the vehicle routes are generated synthetically or are based on real-world data, but in the real world, sensors can fail and usually don't deliver correct data 100% of the time. While this has been addressed in [8], their noise model is very basic, adding a random number to the detected queue lengths.

4 Contributions

The focus of this report is on simulating more realistic sensor data to evaluate the robustness of reinforcement learning agents for traffic signal control (TSC). We use published data on various types of sensor failures [6], [7], [13] to develop a more nuanced noise model that brings the simulations closer to the real world. We then use this data to choose realistic parameters for the noise model and evaluate current agent's performance in environments with such noisy sensor data. This can help cities make informed decisions on which sensors to use when deploying current reinforcement leanring based TSC techniques and enables future research into training more robust TSC agents using reinforcement learning.
We implement our noise model in the [10] library, which provides implementations of several state of the art techniques and supports multiple simulator environments and datasets.

5 On power consumption of RL TSC agents

With the latest large language models costing companies Millions of Dollars every month - even with just inference cost, it's easy to fear that using reinforcement learning agents for traffic signal control could come with significant increases in energy cost. Therefore, we look into power consumption of other devices at signalled intersections.
Old, incandescent traffic lights typically used 60-150 W per light [4], [18], [9]. This was improved upon with the rise of LED traffic lights. These typically only need 10-20 W per traffic light [4], [18], [9]. Through recent innovation, the power consumption of LED-traffic lights could be reduced to only 1-2 W per light [18], [1]. However, given how recent these were published, it's safe to assume that at least currently, these are not very widely used yet.
Most four-way signalled intersections use at least four traffic lights for cars, often eight more for pedestrians and sometimes an additional four to eight for cyclists and four to twelve more for cars. While pedestrian and cyclist lights are usually smaller and dimmer, reducing energy cost, a total of 12 to 20 lights per intersection is not uncommon. Even with the lowest powered traffic signals, this results in a minimum power consumption of 15 to 40 W.
For comparison, a modern laptop CPU at 35 W was able to run the agents presented here at at least 500 times real-time speed or powering many intersections simultaneously, bringing the power consumption per intersection down to well below 1 W. A lower powered device like a Raspberry Pi at a maximum of 6 W [11] would still be sufficient to run these agents. Weighing this against the benefits of lowering travel times, fuel consumption and emissions from stopping vehicles shows that this is an investment that can very quickly offset it's cost in many ways, especially if cameras or other sensors are already installed for other purposes and can be reused for traffic control. Otherwise, these sensor would be the most significant power consumer.

6 Experiment description

6.1 Research Questions

There are several goals for the experiments:

RQ1 Find out how different sensor failure modes affect the performance of TSC agents

RQ2 Find out how different noise levels values affect the performance of TSC agents

RQ3 Check if training with noisy sensor data helps to improve performance compared to training on clean data.

6.2 Experiment setup

We train individual agents for each intersection in the road network. Therefore the state and action spaces will also depend on the intersection and can be different for each agent. Here, we describe how they are constructed.
We then train two reinforcement learning agents: one is trained on clean, undisturbed sensor data, the other is trained on disturbed sensor data with relatively high noise levels. Otherwise, both agents use the same neural network architecture and general agent design described below.
Afterwards we test each agent on different noise settings (see table 6.2) and record their performance. Since the fixedTime agent does not require any sensor data as input, it's performance stays the same in all scenarios.

6.2.1 Dataset: cologne3

This dataset (see Fig. 1) includes three signalled intersections in cologne (Germany) along a corridor with several other roads coming in from the sides. This dataset was included with the LigSignal library [10], who took it from [2] who in turn, to my knowledge, have extracted it from a much larger Cologne dataset created by the DLR [5]. The traffic flow is based on real world traffic-data with the arrival rate over time illustrated in Fig. 2. The phases notably also include signals for right turns, which are absent in many other publications [17], [2] where instead right-turning vehicles have to yield.

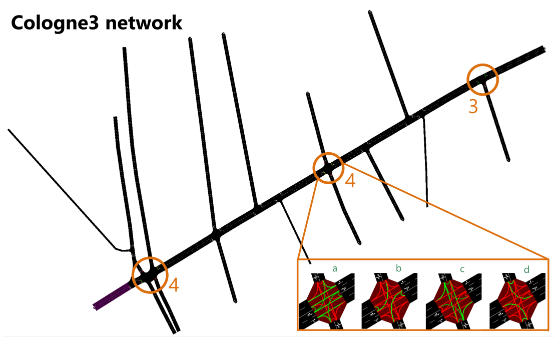

Figure 1: Cologne3 network and phases. All three signalled intersections are highlighted. The numbers show the number of available phases per intersection. The bottom right shows the four phases for the middle intersection. The phases at the other intersections are very similar.

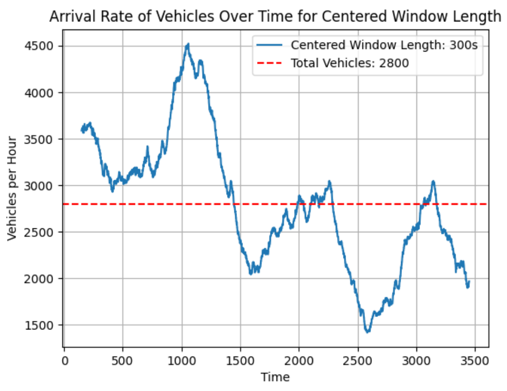

Figure 2: Cologne3 network total vehicle arrival rate over time, smoothed by averaging with a centered moving window of 300s. A total of up to 2800 vehicles enter the network at different traffic flow intensities throughout the simulation interval.

6.2.2 State space

Each observation (state) combines information about the phase, number of incoming vehicles per lane and number of outgoing vehicles per lane. The phase is a one-hot vector encoding all possible phases at the intersection. Transition phases where there are yellow lights or all red lights are not included. Those are automatically generated when switching between phases. In addition to the phase, the state includes the number of vehicles on each incoming and outgoing lane.

6.2.3 Action space

The actions available to the agents correspond to all permissable phases at the agent's intersection. This means that at time t each agent can decide to switch to any other phase or keep the current one. This gives a lot of freedom to the agent by allowing phases to be skipped repeatedly. Note that this is a different action space than what is used in [8], where actions directly correspond to keeping the current phase, switching to the next one or skipping the next phase.

To prevent the agent from switching phases too quickly, an action is only allowed to take actions every 10 seconds (= action interval in table 6.1). Combined with the automatic yellow-phases, this ensures that agents cannot develop unsafe signal patterns.

6.2.4 Rewards

We use a reward based purely on pressure:

$$r_i = -P_i \tag{4}$$

Where i is an identifier of an intersection and $P_i \in \mathbb{R}$ is the pressure of that intersection according to Eq. 3. A high pressure means there are more vehicles incoming than outgoing at the intersection, which can indicate future congestion. A negative pressure indicates more vehicles are on outgoing lanes. Since vehicles don't usually stop on outgoing lanes but can stop on incoming lanes, pressure is, on average over time, expected to be positive. Adding the negative sign makes pressure into a suitable reward to be maximized by the agent.
Since each agent only controls a single intersection and there is no parameter sharing between agents, the rewards are also calculated for each intersection separately and applied only to the corresponding agent.

6.2.5 Agent design

Due to limited availability of code, compatibility with dataset and limited resources, we only test a single reinforcement learning agent here, which is based on *PressLight* [17]. **PressLight agent**
The PressLight agent uses the state and action spaces defined above as inputs and outputs for a neural network. The actions are one-hot encoded such that each neuron in the network's last layer corresponds to one signal phase. Actions are chosen by selecting the output neuron with the highest value. The exact number of inputs and outputs depends on the intersection, as we train one agent per intersection.
Every agent's neural network uses a single hidden layer with 20 neurons, making them fairly small.

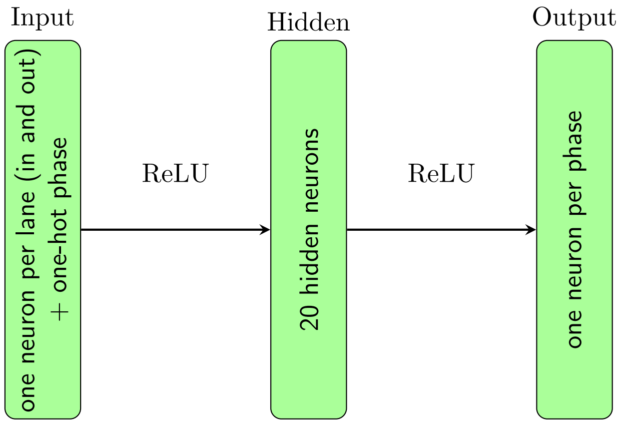

Figure 3: Visualization of the neural network used for the PressLight agents. The network consists only of fully connected layers.

FixedTime agent
This agent uses fixed signal timings. It switches to the next phase every 30s (`t_fixed` in `configs/tsc/fixedtime.yml`).
MaxPressure [15]
MaxPressure is a traditional technique, that greedily selects phases with maximum pressure of the associated traffic movements.

6.2.6 Simulated sensor failures

We simulate three types of faulty sensor data:

1. Complete sensor failure
2. Missed vehicle detection
3. Vehicles detected when none actually passed the sensor

All types of failures can occur with most common sensor types used for traffic detection. All sensors can break, lose power or stop producing outputs for other reasons. This is simulated with failure type 1. Additionally, the detectors' outputs do not always accurately represent the real-world scenario that was observed. Camera-based systems can be prone to getting their field of view blocked, for example by large trucks. Loop detectors can fail to detect vehicles with an insufficient proportion of metal parts. Most sensors can also struggle to accurately count multiple vehicles passing the sensor in quick succession. These failure cases are covered by type 2
Loop detectors can also occasionally misidentify other metallic objects as vehicles. Video
Here, we make a few simplifying assumptions, which are explained in detail in Sec. 7.6.
1 Complete sensor failure
We simulate a complete sensor failure by replacing all outputs of that sensor with 0 whenever that sensor output would be passed to the neural network. Which sensors fail is chosen randomly at the start of each episode and they remain broken for the entire episode. This was chosen because each episode only simulates one hour of real-world time. The failure rate is determined by the parameter `failure_chance`. As a consequence, the sensors were also randomly disabled at the start of each test run. Therefore we repeated every test 15 times to get more representative results.
2 Missed vehicle detection
For all working sensors, we simulate the scenario where a vehicle passes the sensor, but is not recognized as doing so. Given a detection rate (=true positive rate p_{TPR}), this is realized through sampling a binomial distribution, as each detection . The simulator measures the real number of vehicles n_l on every lane l. Before this number is passed to the agent to decide an action, we pass it through the noise filter.

$$n_{l,\text{det}} = \text{Binomial}(n_l, p_{\text{TPR}}) \tag{5}$$

3 Detection without Vehicle present
For all operational sensors, we also simulate the case where a vehicle is detected even though no vehicle actually passed the sensor. Assuming reasonably well calibrated sensors, this should be a rare event. As described in Table 6.1, agent only act in the environment every 10s. Therefore, the sensor outputs are also only read every 10s. This process is modelled using a Poisson distribution under the assumption

$$\mu = \frac{V}{R} p_{\text{FPR}} \tag{6}$$

where V is the expected throughput (see Sec. 6.2.8) during this episode, $R =$ is the total

$$n_{l,\text{det}} = n_{l,\text{det}} + \text{Poisson}(\mu) \tag{7}$$

To estimate the expected throughput in an episode, we differentiate between episodes during training (including both the training and testing episodes) and the test episodes in which the agent's performance is tested under different noise settings.
During the training phase, in the very first episode, the throughput is estimated as 0. This keeps the false positive rate at 0 for the first episode. After this, the expected throughput is always set to the measured throughput of the previous episode. The training process alternates between training episodes (using an ε-greedy strategy) and testing episodes (using a pure greedy strategy).
During the tests for model evaluation, we want all runs to be done with as similar settings as possible and only vary the noise parameters. Therefore, we estimate the expected throughput as the total number of vehicles defined in the simulation's routes file (here, `cologne3_flow.json`). Here, this is set to 2800 vehicles. This is a slight overestimation, since some of the vehicles enter just a few seconds before the end of the simulation. So the actual maximum throughput is slightly lower. Our test results show, that the models can reach a throughput of more than 2600.

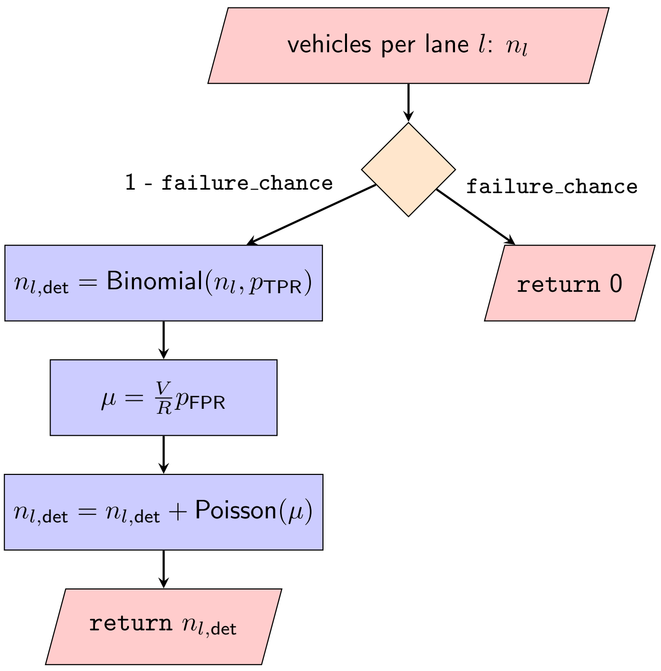
vehicles per lane l: n_l
1 - failure_chance
failure_chance
$n_{l,\text{det}} = \text{Binomial}(n_l, p_{\text{TPR}})$
return 0
$\mu = \frac{V}{R} p_{\text{FPR}}$
$n_{l,\text{det}} = n_{l,\text{det}} + \text{Poisson}(\mu)$
return $n_{l,\text{det}}$

Figure 4: Sensor noise process visualization

6.2.7 Hyperparameters used for training and testing

Here we list all the parameters used for the experiments.

Parameter	**value**	**Simulation setting**	**value**
loss function	MSE loss	timesteps per episode (s)	3600
optimizer	RMSProp	yellow length (s)	5
learning rate	0.001	action interval (s)	10
batch size	64	number of simulation steps	3600
discount factor γ	0.95	number of steps during tests	3600
exploration rate ε	0.1		
epsilon decay	0.995		
epsilon min	0.01		
grad clip	5.0		
replay buffer size	5000		
number of episodes	20		

Table 6.1: Hyperparameters for training and simulation

Parameter	**value**
Sensor failure chances	[0.0, 0.05, 0.1, 0.15]
True positive rates	[1.0, 0.95, 0.8, 0.6]
False positive rates	[0.0, 0.15, 0.3, 0.65]
repetitions per setting	15

Table 6.2: Parameters used for testing

Failure chance 0 means no sensor failures. True positive rate 1 means all vehicles are detected. False positive rate 0 means all detections correspond to actual vehicles. Experiments were performed with every possible combination of these parameters (64 combinations) and each setting was run 15 times with different random seeds to get more representative results using their average values.

6.2.8 Metrics

During training and testing we track several metrics to evaluate each agent's performance. Here, we describe these metrics in detail. All of these metrics are calculated for the whole network. These definitions can be found essentially verbatim in the LigSinal documentation [10, `darl-libsignal.github.io/LibSignalDoc/content/guide/Metrics.html`]. In the travel time definition, I corrected some minor grammatical errors.

Throughput (`throughput`)

> "Number of vehicles that have finished their trips until the current simulation step. A larger throughput means better performance."

Average travel time (`travel time`)

> "The average time that each vehicle spent [...] traveling within the network, including waiting time and actual travel time. [...] Smaller travel time means [...] better performance."

7 Results and discussion

In this section we present the results of the described tests. We first present the impact of each noise parameter for the three tested agents, then a comparison of the agents. Finally, we include plots of metrics tracked during training of the reinforcement learning agents with **RQ3** in mind.
The tests for every parameter configuration were performed 15 times each with different random seeds (0-14). Shown are the averages of these runs. Since this already results in 960 simulations per agent, we did not train the agents multiple times and compared performance between different training runs. Instead, we ensured sufficient training time to reach convergence and verified that the agents converged through the diagrams below.

7.1 Training behavior

To ensure the reinforcement learning agents actually converged, we show the the travel time measured during training. We trained the agent with disturbance for longer because initial tests showed that it took far longer to converge as it has to deal with noisy data. However, given how quickly both agents converged to a good solution, this extra training time likely did not make a significant difference.

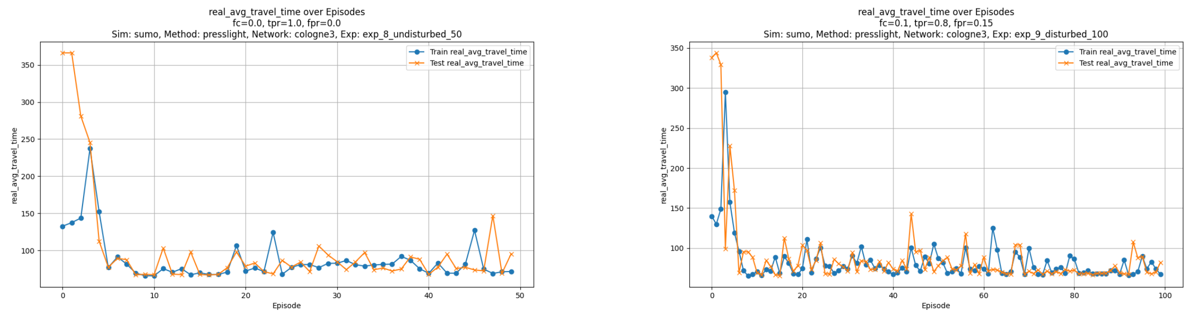

Figure 5: Comparison of travel time during training of RL agents. *Left:* PressLight agent training on clean data for 50 episodes, *Right:* PressLight agent training on disturbed data for 100 episodes.

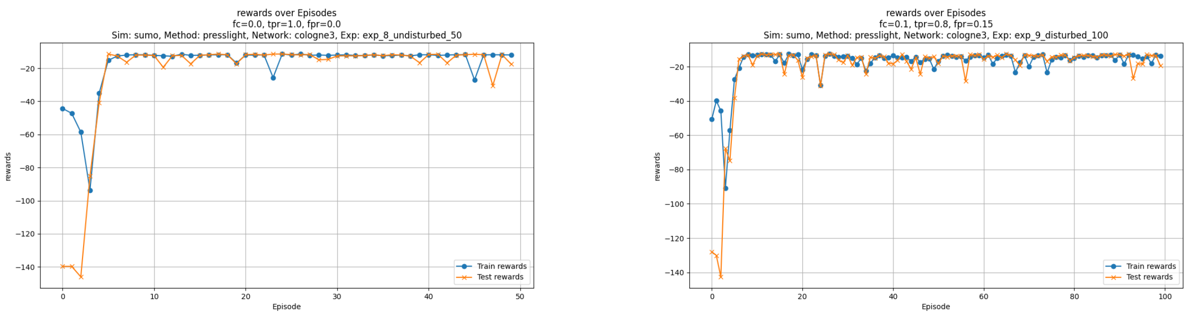

Figure 6: Comparison of rewards during training of RL agents. *Left:* PressLight agent training on clean data for 50 episodes, *Right:* PressLight agent training on disturbed data for 100 episodes.

We see in Fig. 5 and 6 that both agents converge to a similar average travel time of about $70s$ within a few episodes. While this takes slightly longer for the disturbed agent, it still converges very quickly. Earlier iterations took up to 50 episodes to reach a similar travel time. The undisturbed agent always converged similarly fast as here in all tests. Fig. 6 also shows that the disturbed agent had a lot more noisy rewards. Here, it should be noted, that noisy sensor data was also enabled during the test steps shown in this diagram.

7.2 Impact of sensor failure

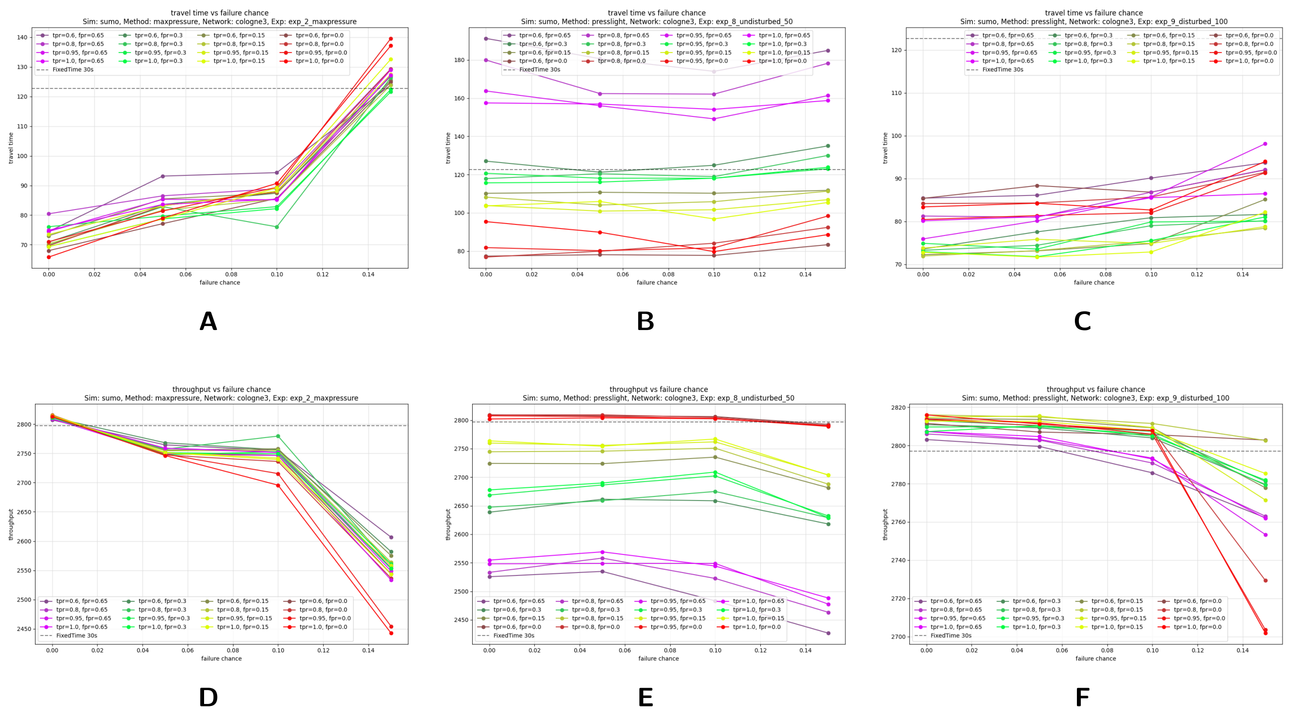

Figure 7: Agent's performance w.r.t. travel time and troughput plotted over failure chances of sensors. A & D show the MaxPressure agent, B & E show the PressLight agent trained on clean data and C & F show the PressLight agent trained on noisy sensor data.

In Fig. 7 A & D, we see that the MaxPressure agent's travel time is quite significantly affected by the failure chance of sensors. A failure chance of 15% leads to almost doubling the average travel time for vehicles. p_{TPR} and p_{FPR} show only small effects, which will be highlighted in the next two sections.
The PressLight agent in Fig. 7 B & E, that was trained on clean data shows a larger range in travel times over the different noise settings, but is mostly invariant to the detectors' failure chance. Here, we already see a doubling in travel time that is caused by high p_{FPR}, which also seems to increase the magnitude of any effects of the failure chance.
Finally, in Fig. 7 C & F, we see the PressLight agent that was trained on noisy sensor data. It shows this agent to be the most robust among all tested agents. It is the only one that remains better than the FixedTime agent at all failure chances (Note the scaling of the plots).

7.3 Impact of true positive rate

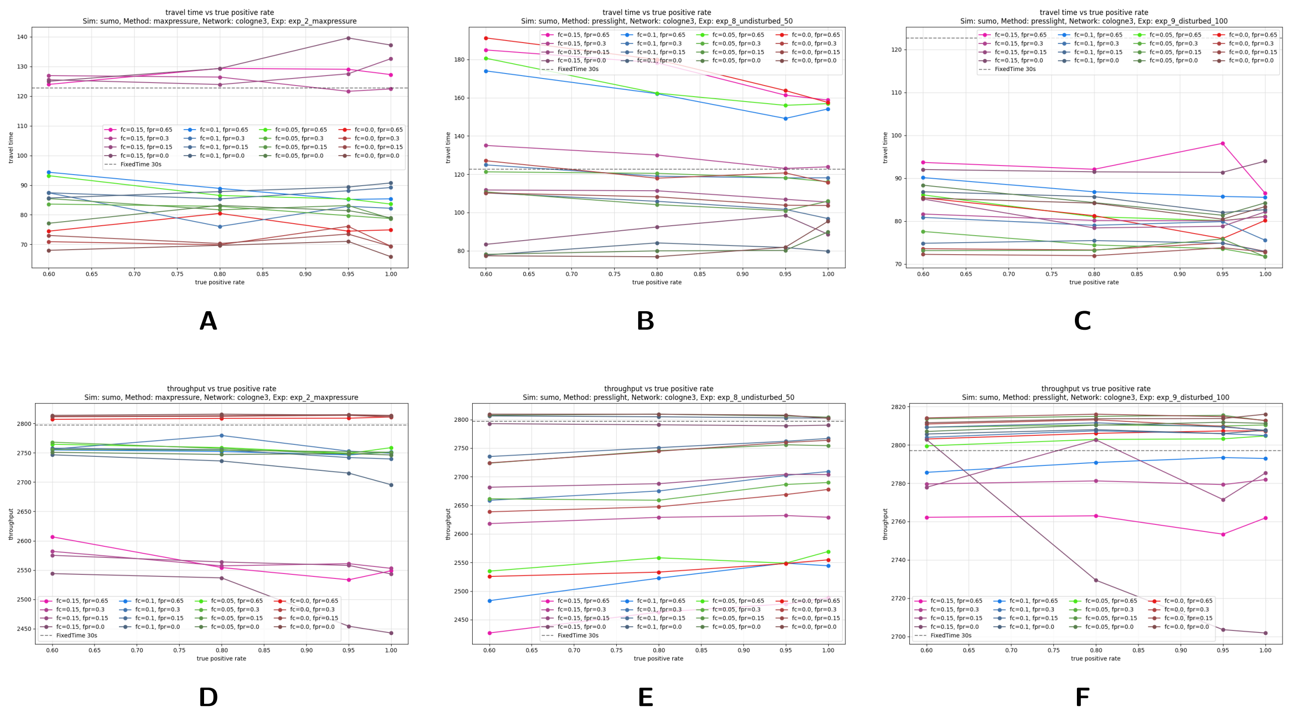

Figure 8: Agent's performance w.r.t. travel time and throughput plotted over true positive rate (detection rate) of sensors. A & D show the MaxPressure agent, B & E show the PressLight agent trained on clean data, and C & F show the PressLight agent trained on noisy sensor data.

Fig. 8 shows that all of the agents are mostly unaffected by the tested ranges for true positive rate. This may be due to the relatively high traffic volume. Even when not detecting 20% of all vehicles, there are still enough incoming to trigger very similar phase changes. Since the rate of missed detections $(1 - p_{\text{TPR}})$ is the same on all sensors, this failure type mostly affects the overall traffic volume that is measured without changing the relative traffic volumes between busy lanes too much. Lanes where there are very few vehicles incoming are less represented in the metrics shown here, since they show an average over all vehicles.
8 D and F show an unusually strong effect when failure chance is high and the true positive positive rate is also high. This combination seems to lower performance of the agents. It may be explained by vehicles on lanes with broken sensors waiting for a long time as they do not get detected. With a lower TPR, the other lanes at the intersection would reach a low enough volume to trigger a phase change more often. We were unable to check whether this is actually the case though.

7.4 Impact of false positive rate

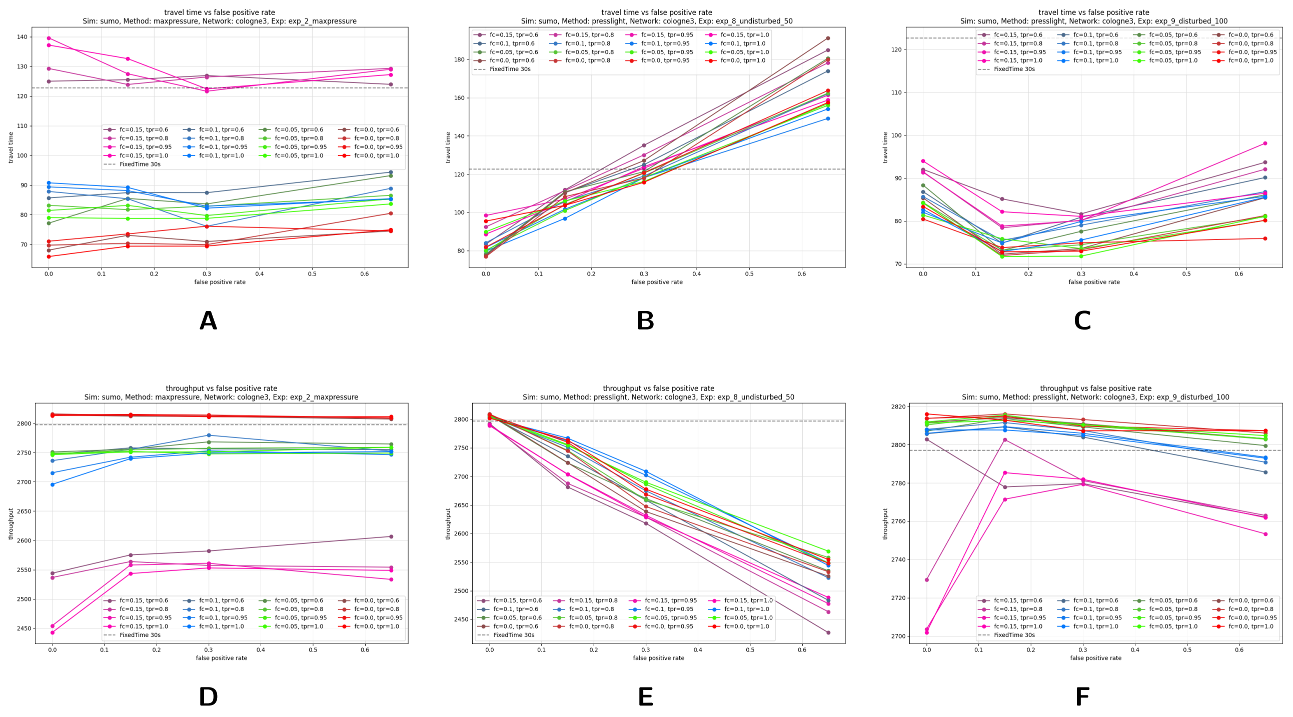

Figure 9: Agent's performance w.r.t. travel time and throughput plotted over false positive rate (misdetection rate) of sensors. A & D show the MaxPressure agent, B & E show the PressLight agent trained on clean data, and C & F show the PressLight agent trained on noisy sensor data.

The false positive rate depicted in Fig. 9 shows the strongest effects for the PressLight agent trained with clean sensor data. Here, a higher rate of false positives triggers a strong and proportional decline in performance, while the other agents are affected far less. Fig. 9 A and D show once again, that the MaxPressure agent is most strongly affected by sensor failures, whereas both noise types have little effect.

7.5 Comparison of agents

Here, we compare the agents' performance directly against each other. For better readability, we only show two different noise settings: One with no noise on the sensor data, the other with noise that is near the upper bound of what is usually expected in the real world according to [6], [7] and [13].

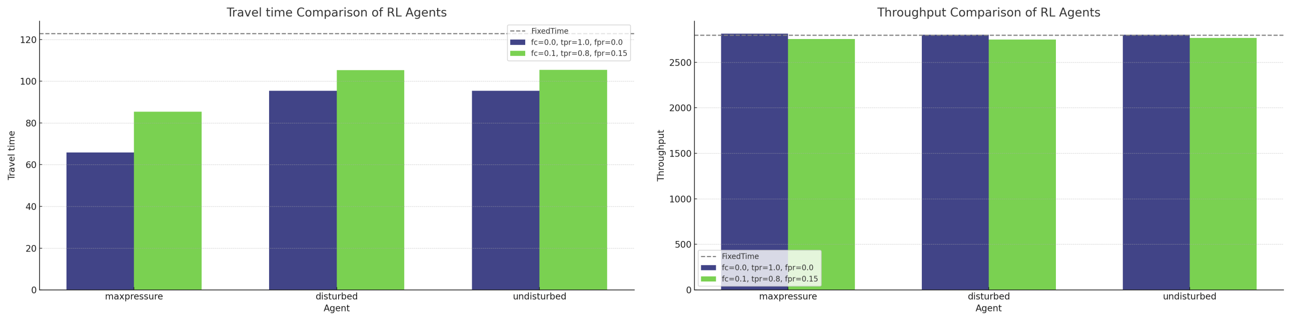

Figure 10: Comparison of all agents with two noise settings and with respect to two metrics: travel time and troughput.

Here, we see MaxPressure performing very well compared to both PressLight agents, which is different from the findings reported in [17, table 4]. This may be explained by the different state definition we used here. The features only include the number of vehicles in each lane, whereas in [17], they used segmented incoming lanes for more granularity, which wasn't possible with the LibSignal library [10] we used here. This difference is illustrated with another learning method in [17, table 5]. The varying arrival rates during our simulation, shown in Fig. 2, would

likely correspond best to the HeavyPeak scenario in that table, suggesting a potentially large performance hit caused by lack of segmentation.
In the earlier results, we saw another important difference between the agents: They react differently to each noise setting. The MaxPressure agent showed a strong negative reaction to sensor failures while being mostly unaffected by the noise added to functional detectors. The PressLight agent trained on clean data

7.6 Known problems

During the definition of the noise model, we made several assumptions. While these seem reasonable, they do not strictly hold true in all situations.
1. Measurements are independent of each other Modelling the added noise with Binomial and Poisson distributions requires the samples to be independent. This is likely not strictly true in the real world. It's possible that specific models or colors of cars cause higher or lower detection rates.
2. No two events occur in the exact same moment The Poisson distribution used for modelling false positives is only suitable for random events, where no two events occur at the same time. Here, the event is "detecting a car, when there is none". Induction loop detectors can indeed detect at most one car at a time, so this assumption holds. However, other detectors like video detectors may not fit this assumption. When given a camera image, and determining how many cars there are, a vision model might be off of the correct number by more than one. This would correspond to two misdetections at the same time. Especially on a busy street or in unfavorable weather conditions, the sensor being off by more than one seems realistic. However, other mitigations can be put in place to solve this problem.
3. Failed sensors are fixed throughout episodes In our model, we made the simplification that if a sensor fails at all during an episode, it does so in all timesteps. Our code currently does not enable sensors breaking or being repaired during an episode. This may be fine for short, one-hour simulations, but could be more inaccurate for longer simulations.
Such failures could be caused by vision systems being occluded by parking trucks or other objects covering the sensor. These types of failure could realistically occur and be fixed even within one hour and are currently not simulated.
4. Incorrect false positive rate when true positive rate < 1 According to def. 2, the false positive rate (misdetection rate) is the number of incorrect detections divided by the total number of detections (both true and false). However, in the calculation 5, we use the number of vehicles actually passing through the network to estimate the number of false positives to add to the clean sensor data. For $p_{\text{TPR}} < 1.$ this overestimates the number of false positives:

$$\text{measured false positives} = \frac{D}{p_{\text{TPR}}} \cdot p_{\text{FPR}} = \frac{W}{D}$$

We decided that this was a sufficiently small error, that we did not re-run the experiments to fix it. Since the overestimation is linear w.r.t. p_{TPR}, the results presented here would only change in magnitude. The overall trends, which provide the most relevant insights, remain unaffected.

8 Further questions

With these tests, we highlighted areas, where future work would be particularly important. In our experiments, the agents had no alternative inputs to compensate for failure of sensors. While training on the noisy data showed some improvements in robustness, it is very likely that the robustness of the learning agents could be further improved by adding information, that is less prone to complete failure - for example the time since the last phase change. Alternatively, introducing memory (e.g. via Long Short Term Memory (LSTM) cells) into the agents could accomplish a similar goal. Since this would be a pure software solution, it could also provide redundancy for the case of incorrect data in other features like the proposed time since the last phase change.

9 Conclusion

We showed in 5, that the power consumption of reinforcement learning agents like the ones used here, can be extremely small compared to that of traffic lights or sensors like cameras.

We then developed a method to model several aspects of potential sensor failures and trained and tested different agents in these noisy environments. Our results align with the conclusion of [13], that infrared detectors would likely provide the most useful sensor data, as it has an extremely low false positive rate. Out experiments have shown the false positive rate of a detector to be much more important than the true positive rate. Compared to loop detectors, these sensors are also much easier to replace when broken, as they can often be accessed without blocking the road.

10 Use of LLMs during the project

During the first few weeks of the project, I used OpenAIs GPT-4 to get a general overview of the field of traffic signal control more quickly than I would have gotten with other sources, often using its web browsing functionality. During the last few weeks, GPT-4 was used to accelerate finding several relevant publications and write up the Bibtex citations more quickly. Additionally, GPT-4 was used to write some of the code for reading the data from the output files and generate the result plots. All of this required many messages and lots of input of both domain, python and software engineering knowledge. All LLM outputs that resulted in further actions were double-checked by myself.

Throughout the programming parts of the project, GitHub Copilot was used as an advanced auto-completion tool, mostly to fill in repetitive code or documentation.

References

[1] *1 Watt Technology*. Accessed: 2023-10-22. URL: https://www.yunextraffic.com/1-watt-technology/.

[2] James Ault and Guni Sharon. "Reinforcement Learning Benchmarks for Traffic Signal Control." In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Ed. by J. Vanschoren and S. Yeung. Vol. 1. Accessed: 2023-10-22. Curran, 2021. URL: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/f0935e4cd5920aa6c7c996a5ee53a70f-Abstract-round1.html.

[3] Chacha Chen et al. "Toward A Thousand Lights: Decentralized Deep Reinforcement Learning for Large-Scale Traffic Signal Control." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.04 (Apr. 2020), pp. 3414–3421. DOI: 10.1609/aaai.v34i04.5744. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5744.

[4] Delaware valley regional planning commision. *Energy efficient traffic signals streetlights*. Accessed: 2023-10-22. Dec. 2010. URL: https://www.dvrpc.org/reports/mit020.pdf.

[5] DLR. *TAPAS Cologne Scenario*. Accessed: 2023-10-27. Unknown. URL: https://sumo.dlr.de/docs/Data/Scenarios/TAPASCologne.html.

[6] Federal Highway Administration. *Title of the Document*. Accessed: 2023-10-21. Oct. 2006. URL: https://www.fhwa.dot.gov/publications/research/operations/its/06139/appendm.cfm.

[7] Federal Highway Administration. *Using GPR to Unearth Sensor Malfunctions*. Accessed: 2023-10-21. Jan. 2011. URL: https://highways.dot.gov/public-roads/januaryfebruary-2011/using-gpr-unearth-sensor-malfunctions.

[8] Soumik Sarkar Kai Liang Tan Anuj Sharma. "Robust Deep Reinforcement Learning for Traffic Signal Control." In: *Journal of Big Data Analytics in Transportation* 2 (3 Dec. 2020). Accessed: 2023-10-22, pp. 263–274. DOI: 10.1007/s42421-020-00029-6. URL: https://link.springer.com/article/10.1007/s42421-020-00029-6.

[9] *LED Traffic Signals*. Accessed: 2023-10-22. URL: https://www.yakimawa.gov/services/streets/led-traffic-signals/.

[10] Hao Mei et al. *LibSignal: An Open Library for Traffic Signal Control*. 2022.

[11] *Power Consumption Benchmarks*. Accessed: 2023-10-23. URL: https://www.pidramble.com/wiki/benchmarks/power-consumption.

[12] Google Research. *Green Light: Using Google AI to Reduce Traffic Emissions*. Accessed: 2023-10-23. 2023. URL: https://sites.research.google/greenlight/.

[13] T Ungureanu et al. *Vergleich der Detektoren für die Verkehrserfassung an signalisierten Knotenpunkten*. German. Vol. 336. Berichte der Bundesanstalt für Straßenwesen. Unterreihe Verkehrstechnik. Main document: https://bast.opus.hbz-nrw.de/files/2447/V336_barrierefrei.pdf, Appendix: https://bast.opus.hbz-nrw.de/files/2447/V336-Anhaenge_barrFrei.pdf, Accessed: 2023-10-23. Fachverlag NW in der Carl Ed. Schünemann KG, Sept. 2020. URL: https://trid.trb.org/view/1753499.

[14] Unknown. "Learning Phase Competition for Traffic Signal Control." In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2019, pp. 3–7.

[15] Pravin Varaiya. "Max pressure control of a network of signalized intersections." In: *Transportation Research Part C: Emerging Technologies* 36 (Nov. 2013), 177–195. DOI: 10.1016/j.trc.2013.08.014.

[16] Hua Wei et al. "CoLight: Learning Network-Level Cooperation for Traffic Signal Control." In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM '19. Beijing, China: Association for Computing Machinery, 2019, 1913–1922. ISBN: 9781450369763. DOI: 10.1145/3357384.3357902. URL: https://doi.org/10.1145/3357384.3357902.

[17] Hua Wei et al. "PressLight: Learning Max Pressure Control to Coordinate Traffic Signals in Arterial Network." In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, 1290–1298. ISBN: 9781450362016. DOI: 10.1145/3292500.3330949. URL: https://doi.org/10.1145/3292500.3330949.

[18] *World's Most Energy-Efficient Traffic Light*. Accessed: 2023-10-22. 2017. URL: https://press.siemens.com/global/en/feature/worlds-most-energy-efficient-traffic-light.

[19] Qiang Wu et al. *Efficient Pressure: Improving efficiency for signalized intersections*. 2021. arXiv: 2112.02336 [cs.LG].

11 Appendix

11.1 Compute Hardware training time

All training and testing was performed on a laptop (Asus Flow X13 (2022) with AMD Ryzen 7 6800hs CPU and 16GB RAM).

Software versions

I used Python version 3.11.3 with the following versions of libraries:

Package Name	Version
gymnasium	0.29.1
lmdb	1.4.1
mpmath	1.3.0
numpy	1.24.1
PyYAML	6.0.1
sympy	1.12

Selbstständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig erstellt und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen, einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.