# Traffic Light Control by Reinforcement Learning

Henrik Sejer Pedersen
Supervisor: Marco Chiarandini

Wednesday 5th December, 2018

# Contents

**Abstract** TODO

# 1 Introduction

With a steady increase in urbanization on top of already existing traffic problems in urban areas, the need for improving the traffic flow becomes ever more apparent. In urban areas, it is likely to be very costly if at all feasible to tear up existing infrastructure to accommodate better roads, so we turn our focus to the dynamic element present in most urban areas today, traffic lights.

Traffic lights provide an opportunity to influence the traffic flow in urban areas utilizing software, be it simple fixed-time, scheduled light changes or sophisticated traffic-reactive systems.

In this project, we look into the online, adaptive optimization strategies through reinforcement learning. Some of the early adaptive approaches SCAT[1] which was implemented in Sydney around 1980 and SCOOT[2], which is in extensive use to this date, focus on the coordination of multiple intersections, often known as *urban traffic control*. In Robertson and Bretherton [2], the coordination serves to shift traffic light cycles such that average queue lengths and vehicle stops are minimized, but still rely on the individual traffic light cycle to be efficient.

In this project, we seek to explore the application of reinforcement learning to control the individual traffic light, using only existing infrastructure.

At a very high level, reinforcement learning consist of an *agent* placed into some *envrionment*, where it takes actions and receive some *reward* based on those actions. Over time, the goal of the agent is to learn which actions to take in what situations to maximize the expected future reward. We will only cover a small part of reinforcement learning in this project, for a wider overview refer to Sutton and Barto [3].

The application of reinforcement learning to the field of traffic light control has been examined many a time[4][5][6][7][8][9][10][11]. However, due to the data made available by swarco Danmark A/S, we can provide further insight into the details of applying reinforcement learning to traffic light control based on existing infrastructure and information.

Thorpe and Anderson [4] showed great potential of reinforcement learning applied to traffic light control with their implementation of the SARSA algorithm on three different complete-knowledge state representations on a grid of 4x4 intersections, approaching the performance of an optimal policy.

Wiering [5] presented three different state representations for the multi-agent problem, where varying degrees of global information was included in the states. He found that more global information did not necessarily mean better performance, but some global information did perform better than none.

Abdulhai et al. [6] implemented Q-learning which is a temporal-difference algorithm, on a single-agent basis, with comments on ideas to improve it for multi-agent use. They used state information which could be estimated from detector data, in the form of queue lengths and elapsed phase time. The reward used is a penalty in the form of total vehicle delay between decision points. Additionally, they made use of a neural-network-like structure (CMAC) to approximate the value function, which provided some generalization and should, to some extent, improve behavior in unobserved states.

Arel et al. [7] used an actual neural network for value function approximation, which should perform even better in unobserved states compared to the CMAC. The state used is also significantly improved, as it incorporates only what they refer to as *relative traffic flow*, calculated for every upstream lane as traffic flow in that lane divided by average traffic flow of all upstream lanes of the intersection. Such a state representation can be implemented using detector data only. The reward is based on average delay, giving a negative reward if an action increases the average delay, but a positive reward if it is decreased.

Genders and Razavi [11] and Touhbi et al. [9] sought out to explore different state- and reward definitions respectively, which we will comment on later in the project, while Yau et al. [10] surveys the use of reinforcement learning in the field of urban traffic control, reflecting on existing research in the field.

# 2 Instance definition

Before going on with reinforcement learning, we cover the underlying simulation with which the reinforcement learning agent will interact.

The simulation implements an intersection located in southern Odense, Denmark, visualized in figure 1. Appendix A contains technical drawings of the intersection.

## 2.1 Network topology

The intersection connects an arterial road (North/South) with a collector street (East) and a gas station (west), all of which have a speed limit of 50 km/h.

### 2.1.1 Induction Loops

In the network, a number of vehicle detectors are present in the form of *induction loops*, visualized in fig. 1 as yellow boxes.

Induction loops are the only way of getting vehicle data from the traffic network, and only very few areas have more sophisticated solutions such as traffic cameras or vehicle radars. We assume that we can retrieve the following information from an induction loop, on a one-second basis:

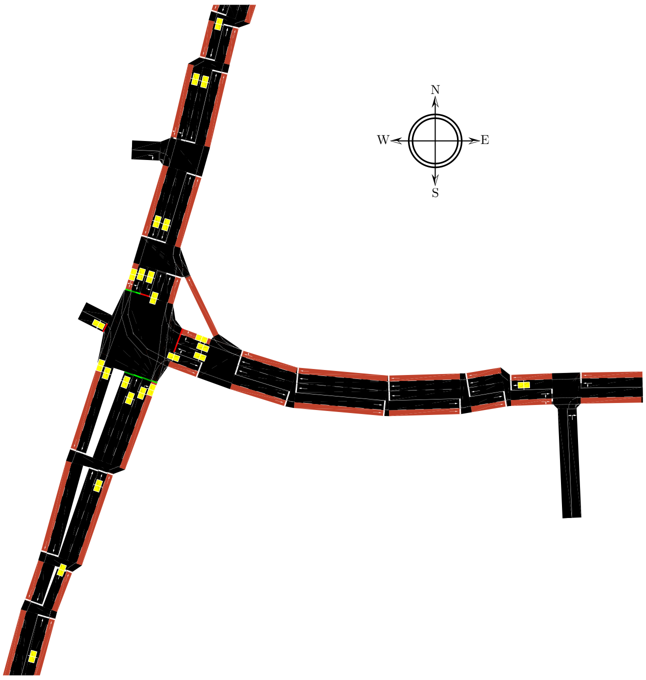


Figure 1: Intersection as implemented in SUMO

- Number of vehicles passed.
- Percentage of time the loop was occupied.

The induction loops in the network cover vehicles quite well, but lack when it comes to bicycles, so we exclude bicycles from the current iteration of the project.

### 2.1.2 Traffic lights

Appendix A also contains a drawing of the traffic light layout at the intersection, from which we can derive the possible resolution of traffic control at the intersection.

The intersection implements three phases. The first phase gives the green light to the south- and north-bound vehicles (A1 and A2), while giving *conditional* green to the left and right turns of those movements (including A1v). The second phase is only activated if vehicles occupy detectors `D23` and `D3` when the first phase goes yellow and gives the green light to southbound traffic (A1), as well as A1v and B1h. The third phase consists of a green light for the west- and east-bound vehicles (B1 and B2), with conditional green for left- and right turns.

## 2.2 Simulator

SUMO, short for *Simulation of Urban MObility*, is a microscopic, well-established general-purpose traffic simulator. It has been around since 2001 and is an open source project. What makes it particularly interesting for this project, is the ability to control elements of the simulation externally, through a well-defined API. With this API, gathering information from the simulation for our agent is made simple, while also providing a way for our agent to control each traffic light.

SUMO uses a directed graph representation to define a traffic network, with some extra information. *Nodes* in the graph are points connected by one or more *edges*. Nodes contain connection data, which is (potentially) a many-to-many mapping of incoming lanes to outgoing lanes. The edges carry the lane information, allowing any number of adjacent lanes (within computational reason) to follow any single edge between two nodes. As such, each edge defines a set of up- or down-stream lanes, possibly consisting of multiple traffic movements.

The primary discrepancy between the actual intersection and the one implemented in this project, is the missing induction loop `D18` located in the center of the intersection. The induction loop is not present in the SUMO implementation as SUMO only allows placing induction loops on lanes.

## 2.3 Vehicle data

Along with technical drawings of the intersection, files describing the traffic flow exist in the form of 5 minutes aggregated readings from the 25 induction loops present in the intersection. To use this data in a meaningful way, we define orderings of the induction loops, each of which defines a route in the network.

We will use linear integer programming to decide which routes vehicles should follow to satisfy these induction loop readings. As the data is non-perfect we propose two different models, the first of which inserts fewer vehicles than the second. In the project we use the second model, to put higher strain on the network.

The first model assumes that the induction loops never miss any vehicles, but may overcount. The second model assumes that the induction loops do not overcount, but may miss vehicles.

Both models give a vehicle count for each route for a single 5-minute interval and are called once for each such period. Solving for shorter intervals increase the network load in the simulation, we use the second model in this project.

### 2.3.1 Overcounts, no misses

Given a set of routes $R$, a set of detectors $D$, a binary route representation as defined below by $B_{ij}$, and upper bounds for *total* number of vehicles passing a detector, defined below by $C_i$. Select the number of vehicles to follow each route, such that the number of vehicles passing each detector is maximized for all detectors, given the single constraint:

- No more than $C_i$ vehicles can pass detector $D_i$, $\forall i \in \{1, 2, \dots, |D|\}$

For convenience we define the sets $I = \{1, 2, \dots, |D|\}$, and $J = \{1, 2, \dots, |R|\}$.

Let $Bij$ be a binary constant such that:

$$B_{ij} = \begin{cases} 1 & \text{if route j passes detector i} \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in I, j \in J$$

Let $C_i$ be an an detected number of vehicles at detector $D_i$, $\forall i \in I$ Let $x_j$ be an integer variable with a lower bound of 0, indicating the number of vehicles following route $j, \forall j\ inJ$

$$\begin{array}{rrcll} \text{Maximize} & \sum_{j \in J} x_j \cdot \sum_{i \in I} B_{ij} & & & \\ \text{s.t.} & \sum_{j \in J} B_{ij} \cdot x_j & \leq & C_i & \forall i \in I \\ & x_j & \in & \mathbb{N} & \forall j \in J \\ & x_j & \geq & 0 & \forall j \in J \end{array}$$

The objective function maximizes the total sum of vehicles passing detectors. The outer sum sums over each route, and the second sum computes the number of detectors in the route from the outer sum.

The first constraint ensures that the sum of vehicles passing a detector does not surpass its capacity.

The second constraint ensures that the number of vehicles entering the simulation is integer.

The Third constraint ensures that the number of cars on each route must be non-negative.

### 2.3.2 Misses, no overcounts

The previous model can with a few changes be modified, such that the vehicle counts act as lower bounds rather than upper bounds. Treating vehicle counts as lower bounds will put a more significant strain on the network, as more vehicles enter the simulation, which is desired.

$$\begin{array}{rrcll} \text{Minimize} & \sum_{j \in J} x_j \cdot \sum_{i \in I} B_{ij} & & & \\ \text{s.t.} & \sum_{j \in J} B_{ij} \cdot x_j & \geq & C_i & \forall i \in I \\ & x_j & \in & \mathbb{N} & \forall j \in J \\ & x_j & \geq & 0 & \forall j \in J \end{array}$$

The objective of the new model is to minimize the number of vehicles passing induction loops, while enforcing that at least the observed number of vehicles pass.

It is important to note that in this model, $B_{i\cdot}$ must contain at least *one* 1 to be feasible for all $i$, whereas this was not the case for the previous model.

# 3 Introduction to reinforcement learning

Reinforcement learning is a machine learning paradigm dedicated to solving sequential decision processes. The definition of a reinforcement learning algorithm given by Sutton and Barto [3, chap. 3], is an algorithm which can solve a specific kind of sequential decision problem, namely those which can be described formally by a *finite Markov decision process.*

## 3.1 Finite Markov Decision Processes

The Markov decision process provides a formal description of sequential decision problems.

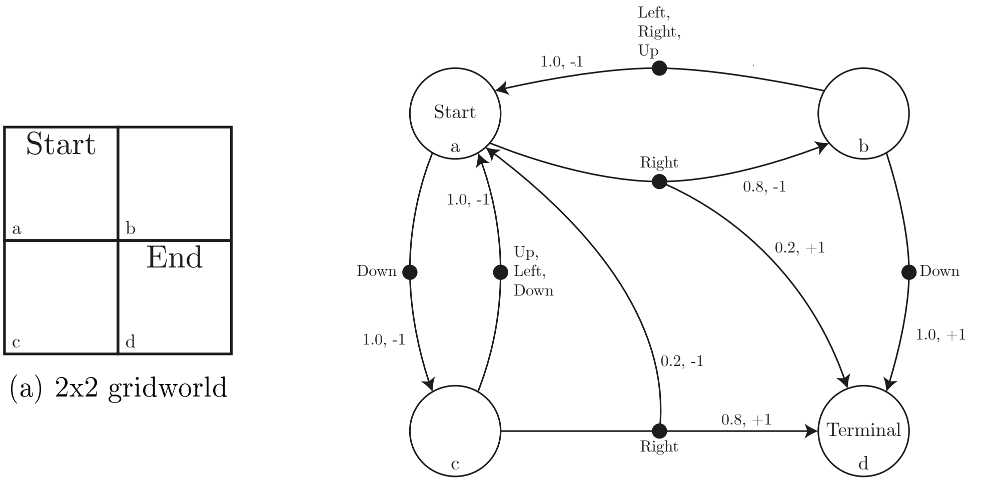
In this project, we define a Markov decision process by a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$, where $\mathcal{S}$ is the finite state-space, $\mathcal{A}$ is the finite action-space, $\mathcal{R} \subset \mathbb{R}$ is the finite reward-space, and the dynamics function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$, defined in eq. (1), describes the conditional probability of entering a state $s' \in \mathcal{S}$ with reward $r \in \mathcal{R}$ given the previous state was $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$ was chosen.

$$p(s', r \mid s, a) = \Pr\{S_t{=}s', R_t{=}r \mid S_{t-1}{=}s, A_{t-1}{=}a\} \tag{1}$$

Some like to include $\gamma$, a *discount factor* in the definition of finite MDP. We, however, leave this definition to the individual reinforcement learning algorithm. The discount factor gives some factor of preference to immediate reward compared to future reward, which can improve learning rate, while ensuring convergence of policies in continous tasks.

The states of a finite Markov decision process must have the *Markov* property. If a state is Markov, i.e., possess the Markov property, the transition to it depend at most on the previous state and action $S_{t-1}$ and $A_{t-1}$, but never earlier states or actions $S_{t-n}$ or $A_{t-n}$ for $n$ greater than 1. An example of a non-Markov state would be when a drone is flying, and the state is a vector containing its coordinates. With such a state representation, any next state will depend on not only the current state and action but also previous states, as the velocity vector is essential to predict the future position of the drone, regardless of action taken.

To give a small example of the Markov decision process, consider the small grid world in fig. 2a. Suppose a robot is initially placed in square $a$, labeled `Start`, and that it has the goal of navigating to square $d$, labeled `End`. If the robot moves off the grid, it is returned to square $a$. Every move the robot makes have a cost of one, except for when successfully navigating to the `End` square, in which case it is rewarded with one point. Finally, suppose that the robot is faulty, and whenever it attempts to go right, there is a 20% probability of moving diagonally down-right. An MDP describing this problem is visualized in fig. 2b.



(a) 2x2 gridworld

(b) MDP visualization of example gridworld with error prone right move

Figure 2: Example environment (a) and MDP describing it (b)

## 3.2 Learning the optimal policy

For any given MDP, the goal is to obtain an optimal *policy* $\pi_*$.
A deterministic policy gives the action to take from state $s$, denoted $\pi(s)$, while a stochastic policy gives the probability of taking action $a$, from state $s$, denoted $\pi(a \mid s)$.

If the complete MDP is known, the optimal policy for the process can be computed without ever interacting with the environment by dynamic programming. Unfortunately, the complete MDP is seldom known, and one must turn to other methods.

The key idea behind reinforcement learning is to compartmentalize into two elements - an agent, and an environment. The agent is able to observe the current state $s \in \mathcal{S}$ and choose some action $a \in \mathcal{A}(s)$ in response. While we can control the agent, the environment is unknown, apart from the assumption that it reacts to the agent's actions by entering some new state $s' \in \mathcal{S}$, and will provide a reward $r \in \mathcal{R}$ depending on the action taken from previous state.

This interaction between agent and environment occurs over a sequence of discrete time steps $t \in \{0, 1, 2, \ldots\}$. At each time step $t$, random variables $S_t$, $A_t$ and $R_t$ indicate the state, action and reward at that time step. These random variables define the *agent-environment loop.*

Figure 3 visualises the agent-environment loop. Initially, the agent is placed in some environment $\mathcal{E}$, from where the agent observes $S_0$. The agent the chooses some action $A_0 \in \mathcal{A}(S_0)$, after the action has been performed, state $S_{t+1}$ and $R_t$ is presented to the agent, this results in a sequence of the form $S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots$

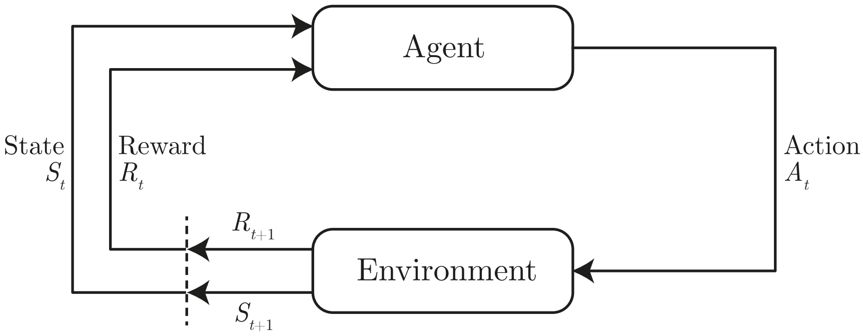


Figure 3: Agent-Environment interface adapted from Sutton and Barto [3, chap. 3]

### 3.2.1 On-policy & Off-policy

### 3.2.2 Reinforcement learning algorithm classes

#### Action-Value

#### Policy-Gradient

#### Actor-Critic

### 3.2.3 Exploration vs Exploitation

A great dilemma in the field of reinforcement learning is the choice of exploration vs. exploitation. When should the agent perform the action it deems best according to previous experience, and when should it explore new actions of which the outcome is less known?

The most straightforward action-selection approach would be greedy action selection. With a greedy action selection, initial action values are essential, as otherwise, the agent may never explore some actions — if there is an action which the agent has not yet attempted, it must see that action as attractive to get some idea of the usefulness of every action.

An improvement to the greedy action selection, which has performed well historically is the $\epsilon - greedy$ approach, where the agent picks a random action with probability $\epsilon$, and a greedy action with probability $1 - \epsilon$.

$$\Pr\{A_t = a \in_R \mathcal{A}(S_t)\} = \epsilon \tag{2}$$

$$\Pr\left\{A_t = \arg\max_a \left[Q_t(S_t, a)\right]\right\} = 1 - \epsilon \tag{3}$$

Where $Q_t$ is the state-action quality measure at time $t$, defined slightly differently between reinforcement learning algorithms.

# 4 Implementation

# Work in progress

As SUMO runs with time steps of one second, we will do the same in this project.

## 4.1 States

## 4.2 Actions

## 4.3 Rewards

# 5 Results

# 6 Conclusion

# TODO: standardize references

## References

[1] A. G. Sims and K. W. Dobinson. The Sydney Coordinated Adaptive Traffic (SCAT) System Philosophy and Benefits. *IEEE Transactions on Vehicular Technology*, 29(2):130–137, 1980.

[2] Dennis I. Robertson and R. David Bretherton. Optimizing networks of traffic signals in real time – the SCOOT method. *IEEE Transactions on Vehicular Technology*, 40(1):11–15, Feb 1991. ISSN 0018-9545. doi: 10.1109/25.69966.

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.

[4] Thomas L. Thorpe and Charles W. Anderson. Traffic Light Control Using SARSA with Three State Representations. Technical report, IBM Corporation, 1996.

[5] MA Wiering. Multi-agent reinforcement learning for traffic light control. In *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*, pages 1151–1158, 2000.

[6] Baher Abdulhai, Rob Pringle, and Grigoris J. Karakoulas. Reinforcement Learning for True Adaptive Traffic Signal Control. *Journal of Transportation Engineering*, 129(3):278, 2003. ISSN 0733947X.

[7] Itamar Arel, Cong Liu, Thomas Urbanik, and A. G. Kohls. Reinforcement learning-based multi-agent system for network traffic signal control. *IET Intelligent Transport Systems*, 4(2):128–135, 06 2010.

[8] Bram Bakker, Shimon Whiteson, Leon Kester, and Frans CA Groen. Traffic light control by multiagent reinforcement learning systems. In *Interactive Collaborative Information Systems*, pages 475–510. Springer, 2010.

[9] Saad Touhbi, Mohamed A. Babram, Tri Nguyen-Huu, Nicolas Marilleau, Moulay L. Hbid, Christophe Cambier, and Serge Stinckwich. Adaptive Traffic Signal Control : Exploring Reward Definition For Reinforcement Learning. *Procedia Computer Science*, 109:513–520, 2017.

[10] Kok-Lim Alvin Yau, Junaid Qadir, Hooi Ling Khoo, Mee Hong Ling, and Peter Komisarczuk. A Survey on Reinforcement Learning Models and Algorithms for Traffic Signal Control. *ACM Computing Surveys (CSUR)*, 50(3):1–38, 2017.

[11] Wade Genders and Saiedeh Razavi. Evaluating reinforcement learning state representations for adaptive traffic signal control. *Procedia Computer Science*, 130:26–33, 2018.

[12] Volodymyr Mnih, Adrià P. Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *CoRR*, abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783.

# A Technical drawings

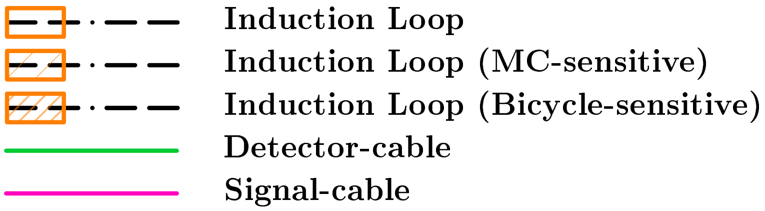

Induction Loop
Induction Loop (MC-sensitive)
Induction Loop (Bicycle-sensitive)
Detector-cable
Signal-cable

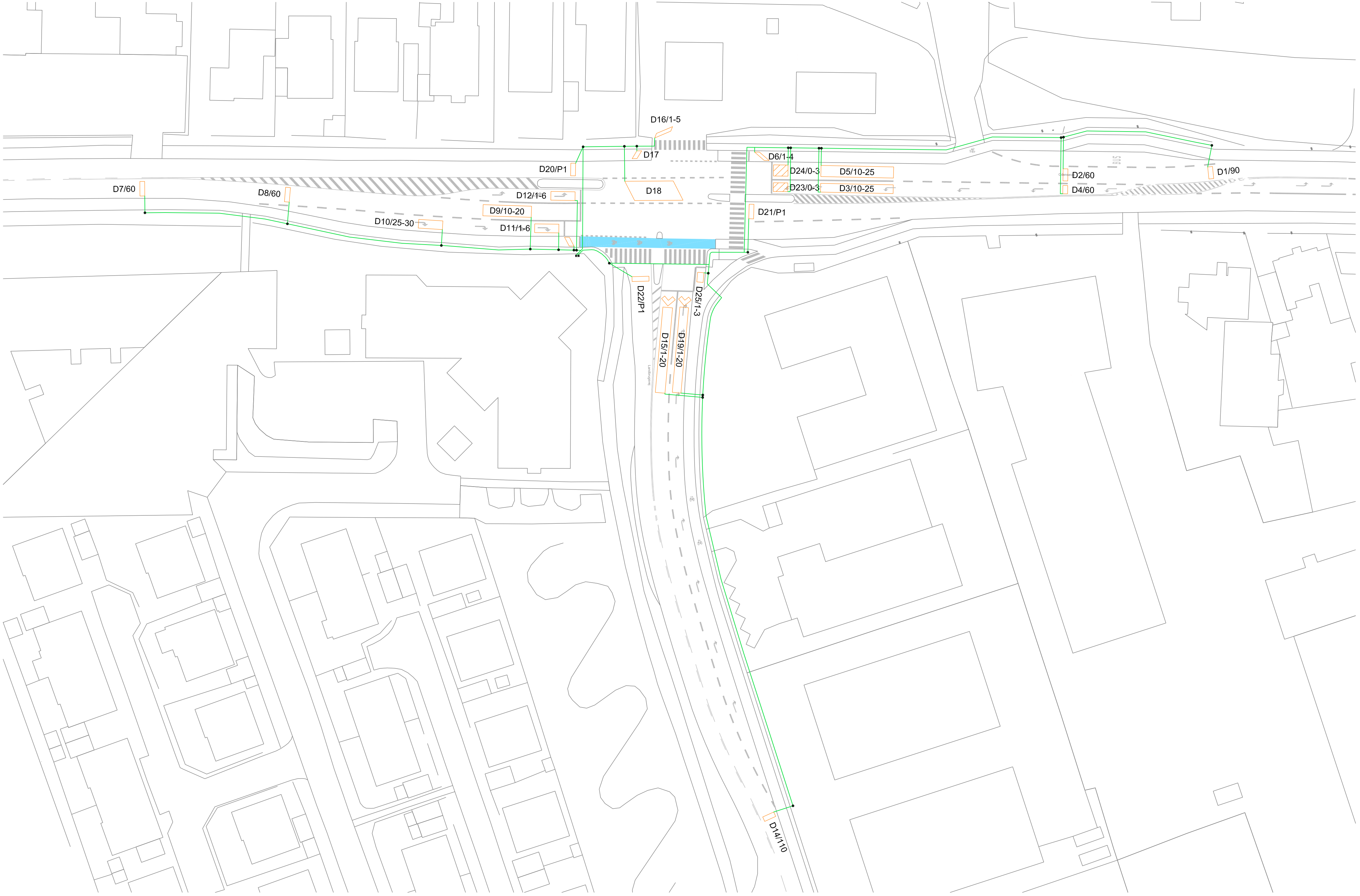
Figure 4: Induction loop drawing legend

| Regular Expression | Explanation |
|---|---|
| [A-B][1-2] | Group of upstream movements on single road |
| [A-B][1-2]Cy | Bicycle movement modifier |
| [A-B][1-2][Vv] | Left-turn modifier |
| [A-B][1-2][Hh] | Right-turn modifier |
| [A-B][1-2](Cy)?([Hh] \| [Vv])? | Non-pedestrian movement strings |
| [a-b][f-g] | Pedestrian light |

Table 1: Explanation of traffic movement strings used in later drawing. Occasionally the movements seem aggregated in the drawing, e.g. B1+B1h; this does not mean that they follow the same light, but rather that multiple lights share the same approximate physical location.

| Rev.1 | Rev.2 | Rev.3 | Rev.4 | Rev.5 | Rev.6 |
|---|---|---|---|---|---|
| | | | | | |

| Rev.7 | Rev.8 | Rev.9 | Rev.10 | Rev.11 | Rev.12 |
|---|---|---|---|---|---|
| | | | | | |



0 2,5 5 7,5 10
1:500

swarco
SWARCO DANMARK A/S

Tonsbakken 16 - 18
DK 2740 Skovlunde
Telefon 36 88 88 88
Telefax 36 88 88 00

Odense Kommune
Odensevej - Landbrugsvej

Oversigtsplan

| PROJ. | TEGN. | DATO |
|---|---|---|
| CJ | FJ | 2017.0105 |

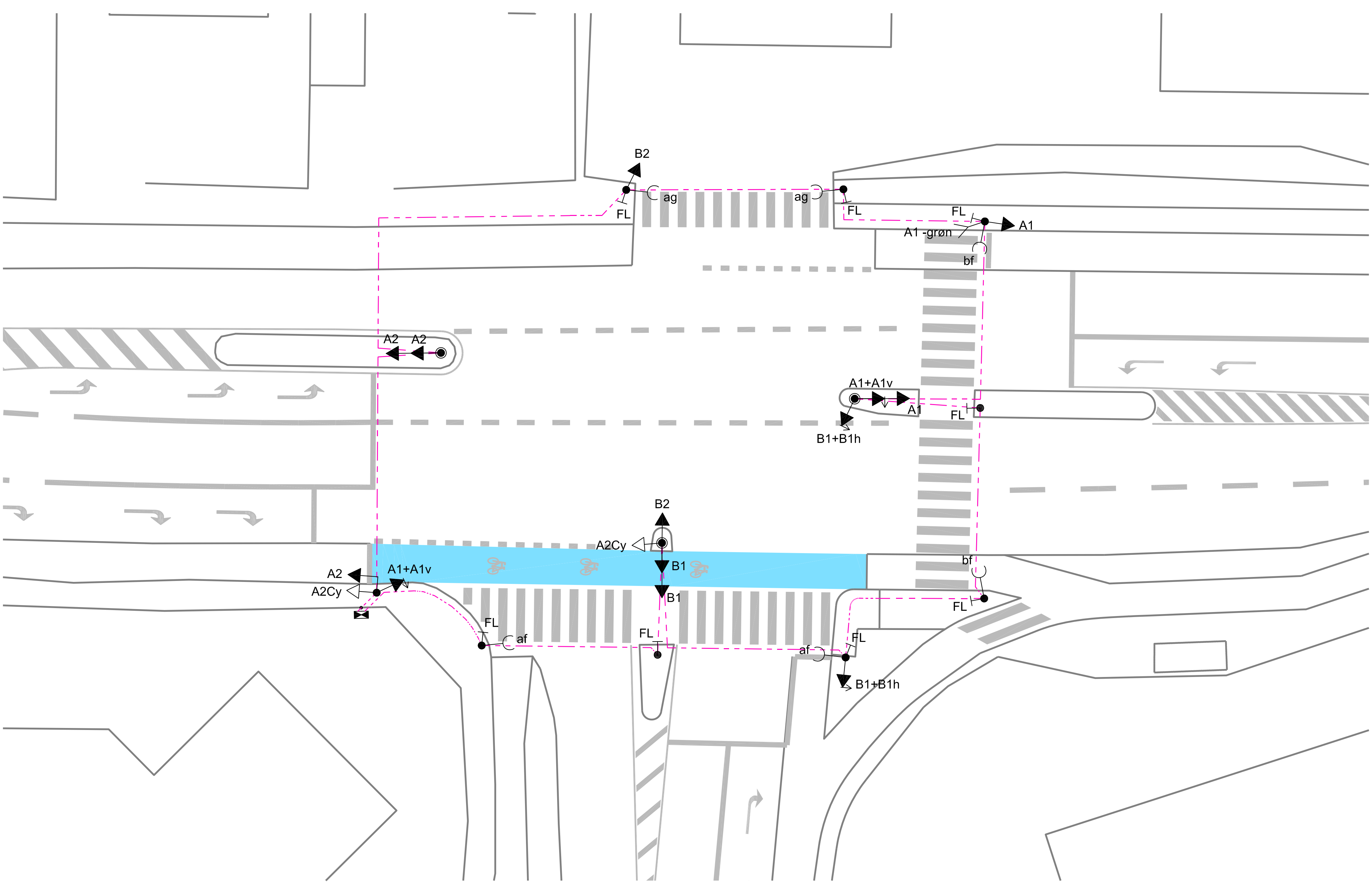
KUNDENR.

TEGNNR.
936.343

REV.

| Rev.1 | Rev.2 | Rev.3 | Rev.4 | Rev.5 | Rev.6 |
|---|---|---|---|---|---|
| | | | | | |

| Rev.7 | Rev.8 | Rev.9 | Rev.10 | Rev.11 | Rev.12 |
|---|---|---|---|---|---|
| | | | | | |



0 2.5 5 7.5 10

1:200

swarco

SWARCO DANMARK A/S

Tonsbakken 16 - 18
DK 2740 Skovlunde
Telefon 36 88 88 88
Telefax 36 88 88 00

Odense Kommune

Odensevej - Landbrugsvej

Oversigtsplan

| PROJ. | TEGN. | DATO |
|---|---|---|
| CJ | FJ | 2017.0105 |
| KUNDENR. | | |
| TEGNNR. 936.343 | | REV. |