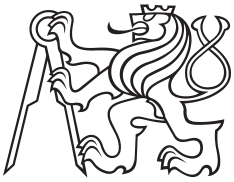
Bachelor Project



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

# Intention Estimation of Traffic Participants Using Bayesian Network

**Antonín Vobecký**

**Supervisor: RNDr. Júlia Škovierová, PhD.**
**Field of study: Cybernetics and Robotics**
**May 2017**

**Czech Technical University in Prague**
**Faculty of Electrical Engineering**

**Department of Cybernetics**

# BACHELOR PROJECT ASSIGNMENT

**Student:** Antonín V o b e c k ý

**Study programme:** Cybernetics and Robotics

**Specialisation:** Robotics

**Title of Bachelor Project:** Intention Estimation of Traffic Participants Using Bayesian Network

**Guidelines:**

The aim of this bachelor thesis is to become familiar with probabilistic programming and Bayesian networks. The thesis will be focused on traffic situations in a simple simulator providing similar data as a self-driving car. The goal of the thesis is to design and implement Bayesian network capable of predicting the intentions of the traffic participants.

**Bibliography/Sources:**

[1] Pfeffer Avi - Practical Probabilistic Programming - Manning Publications, Shelter Island, NY, 2016
[2] Blei David M. - The Basics of Graphical models - Columbia University, 2015

**Bachelor Project Supervisor:** RNDr. Júlia Škovierová, Ph.D.

**Valid until:** the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic
**Head of Department**

prof. Ing. Pavel Ripka, CSc.
**Dean**

Prague, December 19, 2016

# Acknowledgements

First of all, I would like to thank my supervisor RNDr. Júlia Škovierová, Phd. for excellent leadership, immense willingness and patience, and for always great advice.

My thanks also include prof. Ing. Václav Hlaváč, CSc. and Mgr. Radoslav Škoviera, Phd. for their comments and advice that have improved this bachelor thesis.

I would also like to thank Ing. Miroslav Uller for his advice and help.


# Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date .....

.....

signature

# Abstract

This bachelor thesis deals with prediction of behavior of traffic participants using Bayesian networks and is motivated by the European project H2020 *UP-Drive Automated Urban Parking and Driving*. It is important to understand and predict the intentions of traffic participants around an autonomous car. Proper understanding of the situation, together with the intention estimation of individual participants, is important for safety. Many works have already dealt with the topic of understanding and prediction of car behavior in traffic. The prediction of pedestrian intentions is still largely unexplored. This is why we also deal with predictions of pedestrian behavior in the zebra crossing vicinity.

In the introduction, the thesis focuses on the knowledge needed to understand the Bayesian networks. Probabilistic programming is also described. It provides the basis for creating probabilistic models. In this bachelor thesis probabilistic programming language called *Figaro* is used. Next, is a description of a traffic simulator that was created in the Java programming language to test Bayesian network. The last part describes the creation of two Bayesian networks predicting the behavior of traffic participants. The first network is more general and is used with data obtained from the simulator. The second Bayesian network focuses on pedestrian behavior near the zebra crossing and works with real data provided by the *UP-Drive* project.



**Keywords:** Bayesian networks, probabilistic programming; Figaro, autonomous driving, intention estimation.

**Supervisor:** RNDr. Júlia Škovierová, PhD.



# Abstrakt

Tato bakalářská práce se zaobírá predikcí chování účastníků dopravního provozu za použití Bayesovských sítí a je motivována evropským projektem H2020 *UP-Drive Automated Urban Parking and Driving*. Je důležité pochopit a předpovídat úmysly účastníků dopravního provozu v okolí autonomního auta. Správné porozumění situace spolu s predikcí chování jednotlivých účastníků je zásadní s ohledem na bezpečnost. Mnoho prací se již zaobíralo tématem porozumění a predikce chování aut v dopravním provozu, ale předpovídání úmyslů chodců je stále velmi neprozkoumané. To je také důvod, proč se v této práci mimo jiné zabýváme právě předpovědí chování chodců v blízkosti přechodů. V úvodu se práce soustředí na teoretický základ potřebný pro pochopení problematiky Bayesovských sítí. Dále je popsáno pravděpodobnostní programování, které poskytuje prostředky pro vytváření pravděpodobnostních modelů. V této bakalářské práci je použit pravděpodobnostní programovací jazyk s názvem *Figaro*. Následuje popsání simulátoru dopravních situací, který byl vytvořen v programovacím jazyce Java za účelem testování Bayesovských sítí. V poslední části je popsáno vytvoření dvou Bayesovských sítí zaměřených na predikci chování účastníků dopravního provozu. První síť je více obecná a je použita s daty získanými ze simulátoru. Druhá síť se zaměřuje na chování chodců v blízkosti přechodů a pracuje s reálnými daty poskytnutými z projektu *UP-Drive*.



**Klíčová slova:** Bayesovské sítě, pravděpodobnostní programování, Figaro, autonomní řízení, odhadování úmyslů.

**Překlad názvu:** Odhadování úmyslů účastníků silničního provozu pomocí Bayesovských sítí —

# Contents

# Figures

# Tables

# Introduction

## Motivation

The research in the area of automated driving research is currently very popular in academia and also in industrial research. A self-driving car has has the potential of improving safety, reducing congestion, lower emissions and greater mobility [4]. However, automated vehicles must interact with other traffic participants such as human-driven vehicles and pedestrians.The correct estimation of the expected behaviors of other traffic participants is essential.

There were enormous advances in the research of automated driving in the last two decades. One of the first attempts in the area of automated driving was the Eureka PROMETHEUS project in the 1990s which dealt with automated lane keeping and cruise control. In this project, the tour from Munich, Germany to Odense, Denmark was made with 95% of automated lane keeping [2]. In a similar project in the USA called 'No hands across America', the tour was completed with 98% of automated lane keeping [10].

The ever-growing research area of autonomous cars unveils a large amount of new challenges. Scenario understanding of surrounding environment is one of these challenges and is crucial for proper management of autonomous car.

Despite the progress in autonomous cars, many unsolved problems remain, including driving in highly populated areas. This bachelor thesis contributes to *UP-Drive* project, which addresses these issues. In this thesis, we consider traffic situations from the autonomous' vehicle point of view.

Scenario understanding of surrounding environment is one of the arisen challenges and is crucial for proper management of autonomous car. A lot of research has been done to predict car intentions and behaviors and a very little to predict intentions of pedestrians. That is one of the reasons why we deal with the prediction of pedestrian intentions in the vicinity of zebra crossings in this thesis. Another reason is, that the safety of the traffic participants and especially pedestrians in the complicated traffic situations is crucial. Based on the [14], 1.2 million people in the world die in traffic accidents each year. With improved intention estimation of pedestrian intentions capable of predicting pedestrian intentions more accurately, this number can be reduced.

## Task Formulation

This thesis deals with topic of Intention Estimation of Traffic Participants Using Bayesian Network.

Bayesian networks have not been part of any mandatory lectures. The first goal is to acquire knowledge needed for understanding given topic. This involves mostly the knowledge of probability theory and graph theory.

The second goal is to familiarize probabilistic programming. Probabilistic programming is a tool for designing probabilistic models, Bayesian networks in our case.

For the purpose of development and testing of designed Bayesian network, it is essential to create a simulator providing similar data as self-driving car. Creating a simulator like that is therefore another goal.

The main goal of this thesis is to design and implement Bayesian network capable of predicting the intentions of the traffic participants.

# Chapter 1

# Bayesian networks

As this Bachelor thesis is closely related to the topics of probability, it is necessary to describe basic used rules of probability. We begin in Section 1.1 with describing conditional independence with The law of total probability and Chain Rule. In the following sections we describe Markov Chain (Section 1.2), Directed Graphical Models (1.3.1) and Bayes ball algorithm and $d$-separation (Section 1.4). These topics are closely related to the Bayesian Networks described in section 1.6.

## 1.1 Conditional independence

Let us first recall the definition of independence of two random variables $x_A$ and $x_B$. If these variables are independent, the following statement holds true:

$$p\left(x_A, x_B\right) = p\left(x_A\right) p\left(x_B\right)$$
$$p\left(x_A | x_B\right) = p\left(x_A\right)$$
$$p\left(x_B | x_A\right) = p\left(x_B\right)$$

The fact that variables $x_A$ and $x_B$ are independent is often denoted as $x_A \perp x_B$.
Likewise, for two random variables $x_A$ and $x_B$ being independent given $x_C$ holds true:

$$p\left(x_A | x_B, x_C\right) = p\left(x_A | x_C\right)$$
$$p\left(x_B | x_A, x_C\right) = p\left(x_B | x_C\right)$$

This conditional independence is often denoted as $x_A \perp x_B | x_C$

### 1.1.1 The law of total probability

Suppose we have a set of mutually exclusive and exhaustive random events (variables) $X_1, ..., X_N$, where $p(X_i) \neq 0 \ \forall i$. For an event $Y$ of the same space, the following equation holds [5]:

$$p\left(Y\right) = \sum_{i=1}^{N} p(Y | X_i) p\left(X_i\right) \tag{1.1}$$

### 1.1.2 Chain Rule, factorization of a joint probability distribution

First of all, we need to define joint probability distribution (JPD). JPD is a probability distribution for two or more random variables.
Chain rule, also sometimes called product rule, permits calculation of the joint probability distribution of a set of random variables using only conditional probabilities [11]. In other words, the Chain rule is used to express the joint probability as the product of the conditional probabilities [6].

First we need to recall the definition of conditional probability:

$$p\left(x_1, x_2\right) = p\left(x_1 | x_2\right) p\left(x_2\right) \tag{1.2}$$

Using Equation 1.2, we can write for a set $X_1, ..., X_N$ of random variables:

$$p\left(X_1, ..., X_N\right) = p\left(X_1 | X_2, ..., X_N\right) p\left(X_2, ..., X_N\right) \tag{1.3}$$

Expanding Equation 1.3, we get for random variables $X_1, X_2, X_3$:

$$p\left(X_1, X_2, X_3\right) = p\left(X_1 | X_2, X_3\right) p\left(X_2 | X_3\right) p\left(X_3\right) \tag{1.4}$$

As we might see from Equations 1.3 and 1.4, we can generalize this process by *chain rule*:

$$p(X_1, \ldots, x_n) = \prod_{i=1}^{N} p\left(X_i | \bigcap_{j=1}^{i-1} X_j\right) \tag{1.5}$$

## 1.2 Markov Chain

Markov Chain is the basic kind of a dynamic system - the system, in which the state varies over time and those states are dependent at different times[9, p. 231]. The state of Markov Chain is represented by one variable. This state depends directly only on the state at the previous step/time and not on any other prior states. We can see that on the following example in Figure 1.1:

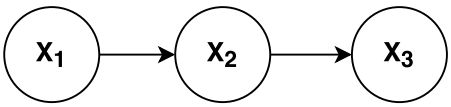



**Figure 1.1:** 3-state Markov Chain.

As we can see, state $X_2$ depends directly only on state $X_1$ and state $X_3$ depends directly on state $X_2$ and not on state $X_1$. We can image this situation as follows:

- state $X_1$ is *past*
- state $X_2$ is *present*
- state $X_3$ is *future*

We can say that *future* ($X_3$) is conditionally independent on *past* ($X_1$) given *present* ($X_2$). The same statement holds true for any states in the future. This statement is an example of the use of the Markov Assumption.

### 1.2.1 Markov Assumption

A dynamic probabilistic model satisfies the Markov assumption if a state at any time point depends only on the directly previous state; the state at any time point is conditionally independent of all earlier states given the directly previous state [9, p. 232].

The directed acyclic graph is a finite graph with no directed cycles. It consist of nodes and edges. Each edge is directed from one node to the another.

We may rewrite this as well in a following manner: *Probability of each variable in the directed acyclic graph (section 1.3.1) is conditionally independent of its nondescendants given its parents.*

## 1.3 Graphical models

A graphical model is a tool that is used to visually illustrate and work with conditional independences among variables in given problem [12]. A graph is composed of a set of nodes (which in graphical models represent random variables) and a set of edges. Each edge interlaces two nodes, and an edge can have an optional direction assigned to it, so it can be oriented or not [6]. If edges of the graph have assigned directions, we call this graphical model a *directed graphical model*.

### 1.3.1 Directed graphical models

We have already seen one graphical model - Markov Chain. Bayesian network is another type of the Directed graphical model.

The Directed graphical model is a directed acyclic graph, where nodes are random variables $x_1, ..., x_N$ and edges represent directed relations (parentage) between nodes. Parents of $x_i$ will be denoted as $\pi_i$. Let us show an example in Figure 1.2:

Random variables in this example are $x_1, x_2, ..., x_5$. Parent of $x_5$ is $x_3$, therefore $\pi_5 = \{x_3\}$. This graph defines a factorization of the joint distribution in terms of conditional distributions $p(x_i|\pi_i)$.

$$p\left(x_1, x_2, x_3, x_4, x_5\right) = p\left(x_1\right) p\left(x_2|x_1\right) p\left(x_3|x_1\right) p\left(x_5|x_3\right) p\left(x_4|x_2, x_3\right)$$

This factorization of the joint distribution looks like this in general:

$$p\left(x_{1,...,N}\right) = \prod_{i=1}^{N} p\left(x_i|\pi_i\right)$$

As we might see, this is an implementation of Markov Assumption and the chain rule (Equation 1.5).

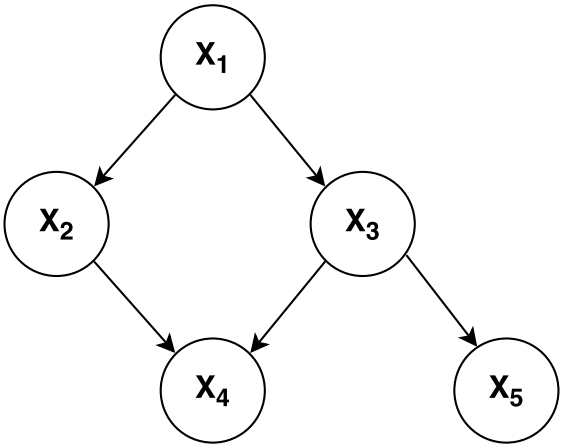



**Figure 1.2:** The example of the directed acyclic graph with four variables.

## 1.4 *d*-separation and Bayes Ball algorithm

We describe two related topics in this section. We will start with Bayes ball algorithm and explain the *d*-separation in the second part of this section.

### 1.4.1 *d*-separation

First thing to note in this section is that *a node separates parents from their ancestors*. Using the example from Figure 1.1,

$$p\left(X_1, X_2, X_3\right) = p\left(X_3|X_2\right) p\left(X_2|X_1\right) p\left(X_1\right). \tag{1.6}$$

As we can see for this sequence, $X_3$ is independent of $X_1$ given $X_2$ since parent $\pi_3$ of node $X_3$ is $X_2$. To prove this, we can show that $p(X_1|X_2, X_3)$ is equal to $p(X_1|X_2)$ and therefore confirms this independence. From the definition of conditional independence, we can write:

$$p\left(X_1|X_2, X_3\right) = \frac{p\left(X_1, X_2, X_3\right)}{p\left(X_2, X_3\right)}$$

Now we can rewrite the nominator using Equation 1.6. We may rewrite the denominator as $p\left(X_3|X_2\right) p\left(X_2\right)$ and then expand it further by using the law of total probability (Equation 1.1), so $p\left(X_2\right) = \sum_{X_1} p\left(X_2|X_1\right) p\left(X_1\right)$. We get:

$$p\left(X_1|X_2, X_3\right) = \frac{p\left(X_3|X_2\right) p\left(X_2|X_1\right) p\left(X_1\right)}{p\left(X_3|X_2\right) \sum_{X_1} p\left(X_2|X_1\right) p\left(X_1\right)}.$$

In the next step, we can divide both the nominator and the denominator by $p\left(X_3|X_2\right)$. Now we may rewrite the remaining denominator again using Equation 1.1 as $p(X_2)$ and see that the remaining nominator $p(X_2|X_1)p(X_1)$ is equal to the joint probability $p(X_1, X_2)$.

$$p\left(X_1|X_2, X_3\right) = \frac{p\left(X_1, X_2\right)}{p\left(X_2\right)}$$

Because $p\left(X_1, X_2\right) = p\left(X_1|X_2\right) p\left(X_2\right)$, we can see, that the result is equal to

$$p\left(X_1|X_2, X_3\right) = p\left(X_1|X_2\right)$$

From this we assert that no other independence hold and so it is not true that $X_1$ is independent of $X_3$. We can interpret this situation as a graph separation as well. Looking at it this way, $X_2$ separates $X_1$ and $X_3$.

Let us show another example. Assume following graph in Figure 1.3. It shows a *tree graph*. *Tree* is a connected acyclic graph.

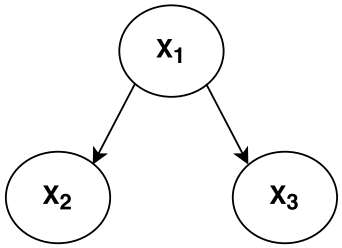



**Figure 1.3:** Tree graph with 3 nodes.

We have

$$p(X_1, X_2, X_3) = p(X_1)p(X_2|X_1)p(X_3|X_1) \tag{1.7}$$

In this example, $X_2$ is independent of $X_3$ given the node $X_1$. To show that:

$$p\left(X_2, X_3|X_1\right) = \frac{p\left(X_1, X_2, X_3\right)}{p\left(X_1\right)}$$

Now we substitute the nominator by Equation 1.7 and get the following:

$$p\left(X_2, X_3|X_1\right) = \frac{p\left(X_1\right) p\left(X_2|X_1\right) p\left(X_3|X_1\right)}{p\left(X_1\right)} = p\left(X_2|X_1\right) p\left(X_3|X_1\right) \tag{1.8}$$

The result of Equation 1.8 can be written as $x_2 \perp x_3|x_1$.

Once again this can be viewed as a graph separation where $X_1$ separates $X_2$ and $X_3$. Let us show it on simple example in Figure 1.4, where we take $X_1$ as the *cold* which causes the *sneezing* ($X_2$) and the *runny nose* ($X_3$):

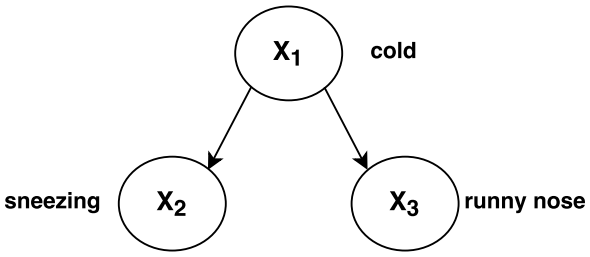



**Figure 1.4:** Example of tree graph.

We will refer to the probability of *sneezing* ($X_2$) as $p(S)$, probability of the *cold* ($X_1$) as $p(C)$ and probability of the *runny nose* ($X_3$) as $p(R)$.

If you are sneezing, it makes it more probable that you have a runny nose too $\Rightarrow p(R|S) > p(S)$. We see from this, that $p(R)$ and $p(S)$ are not independent.

If we know that the person has a cold and we want to know whether the person is sneezing, it does not matter anymore whether she has a runny nose as long as we know that she has a cold. So the fact that we know that the person has a runny nose does not make it more probable that it is sneezing. This means that *sneezing* is independent of *runny nose* given *cold* $\Rightarrow p(S|R,C) = p(S|C)$. This is another example of Markov Assumption introduced in Section 1.2.1.

As a last example shown in Figure 1.5 we have a V-structure, sort of a reversed tree from Figure 1.3:

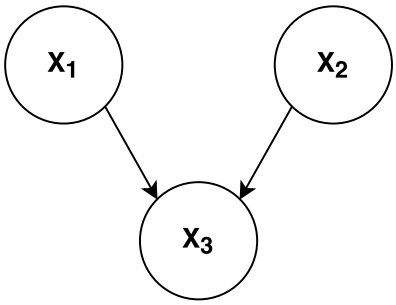



**Figure 1.5:** V-structure.

Here, the joint probability is:

$$p\left(X_1, X_2, X_3\right) = p\left(X_1\right) p\left(X_2\right) p\left(X_3|X_1, X_2\right)$$

In this case, the only independence is the independence between $X_1$ and $X_2$. These two variables do not have to be independent in the case we observe the value of $X_3$. Let us show it in the following famous example in Figure 1.6:

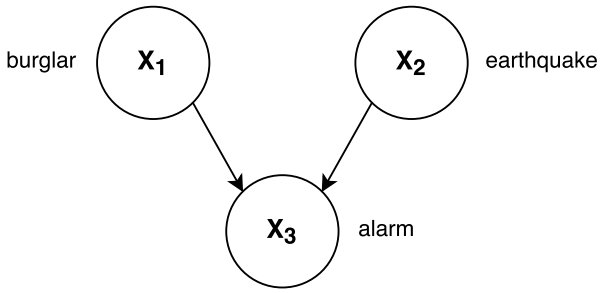



**Figure 1.6:** Example of V-structure.

Marginally, earthquake and burglary are independent (burglar will most probably not cause an earthquake and vice versa). But if we observe the alarm has gone off, than knowing whether there was an earthquake has an effect on the probability that the alarm has gone off because of the burglar, so these two variables are not independent given their common cause.

We need to note, that the fact that the alarm went off does *not* create a *causal* relationship between burglar and earthquake. It creates a *probabilistic/statistics* relationship between these two variables.

Let us have a look at the *d-separation* itself now. Let us have two variables $X$,$Y$ and a set of variables $V$. We can say that $V$ *d-separates* $X$ and $Y$ if *every* undirected path between $X$ and $Y$ is *blocked* by $V$ [3]. The path from $X$ to $Y$ is blocked by $V$ if there is a node $W$ on the path such that either:

1. $W$ has converging arrows along the path $\rightarrow W \leftarrow$ and neither W nor its descendants are observed (in $V$), or
2. W does not have converging arrows along the path ($\rightarrow W \rightarrow$ or $\leftarrow W \rightarrow$) and $W$ is observed ($W \in V$).

Examples of these 'blockings can be seen in Figure 1.7. Examples in these sections were taken from [7].

### 1.4.2 Bayes Ball algorithm

As written in [3], we may explain and understand the Bayes Ball algorithm in the following manner:

It is a 'reachability algorithm'. It means that we imagine a ball starting to bounce at one variable while given some set of observed variables $K$. If the ball can reach the other variable, it means that they are not conditionally independent given the set $K$. Depending on the direction, the ball came from and the type of node, the ball can behave like this:

- the ball can *pass through* from a parent to all children or from a child to all parents,
- *bounce back* from any parent to all parents or from any child to all children,
- or be *blocked*.

This behavior is contained in ten rules of this algorithm as shown in [8]. It states that 'An undirected path is active if a Bayes ball travelling along it never encounters the "*stop*" symbol: $\rightarrow |$'. These rules are shown in Figure 1.7. Symbol $\rightarrow |$ stands for the blocked path, symbol $\rightarrow$ for unblocked path (the ball can pass through) and the 'return arrow' in the top right corner of the example means bouncing back. If the circle (variable) is *gray*, it means that this variable is observed.

## 1.5 Conditional probability table (CPT)

The conditional probability distribution is the probability of the assignment to a variable, given known assignments for another variable(s). $\mathcal{P}(X|Y)$ is the probability of every possible assignment to $X$, for every possible assignment to $Y$, for discrete variables.

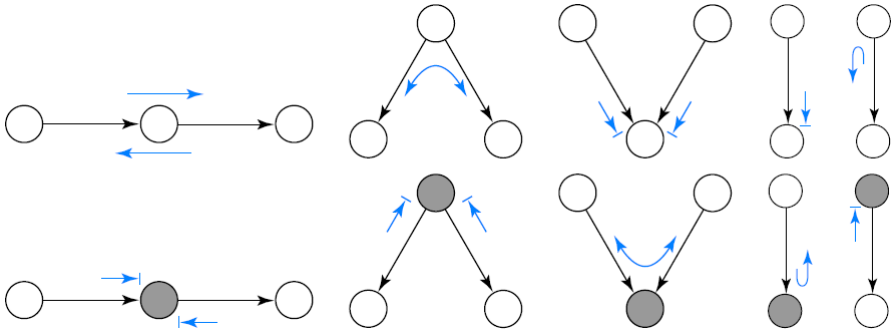


**Figure 1.7:** Rules of flow and blocation for Bayes ball algorithm [8].

The example of the such conditional probability is presented in Table 1.1. The table shows the CPT for the conditional probability $\mathcal{P}(X|Y,Z)$, where $Y,Z$ is the evidence ($Y$ = *there was a burglary*, $Z$ = *there was an earthquake*), and $X$ is the requested result describing the probability that the alarm has gone off.

| burglar | earthquake | p(alarm) |
|---|---|---|
| true | true | 0.9 |
| true | false | 0.9 |
| false | true | 0.3 |
| false | false | 0.05 |

**Table 1.1:** Conditional probability distribution

We will use this type of tables in our Bayesian networks to model conditional probability distributions over variables given its parents.

### 1.5.1 Use of CPT with continuous distributions

We can use continuous distributions as an input of a CPT as well. Such a distribution is shown in Figure 1.8. This distribution indicates pedestrian closeness to the zebra crossing. It can have value in range $\langle 0, 1\rangle$. In this case we need to *sample* this distribution to obtain either *true* or *false* value.

Each sample is a possible world, and the probability of generating a particular possible world should be equal to the probability of that possible world. For example, if the closeness has value 0.72 and we use 100 samples, we get approximately 72 samples with value *true* and 28 samples with value *false*. In the case we have more input distributions, we *sample* these distributions too. We obtain number of samples that match rows in the CPT and get a probability of each sample. The resulting probability is an average of samples probabilities.

This procedure is used with all the conditional probability tables in this thesis.

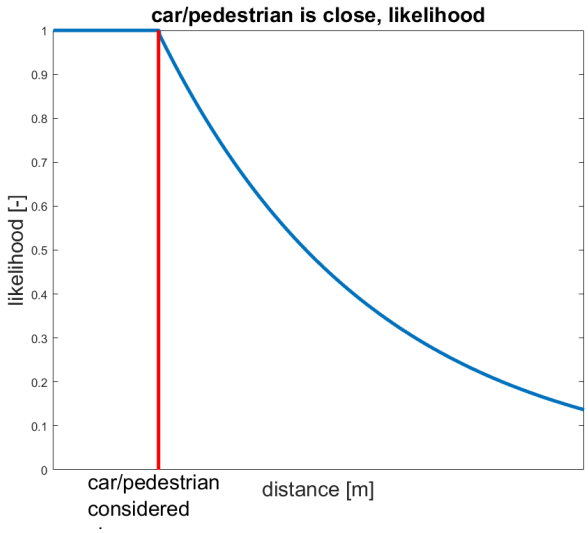



**Figure 1.8:** Distribution of likelihood given pedestrian distance to the zebra crossing.

## 1.6 Bayesian network

Bayesian model, also known as belief network or probabilistic directed acyclic graphical model is another type of the directed graphical model (Section 1.3.1). We can define Bayesian network in a similar manner as in [9, p. 140]:

Bayesian network is a representation of a probabilistic model consisting of three components:

- A set of variables with their corresponding domains
- A directed acyclic graph in which each variable is a node
- For each variable, a conditional probability distribution over the variable, given its parents

Edges in the directed acyclic graph represent conditional dependencies between connected variables. Therefore those nodes that are not connected are assumed to be conditionally independent.

### 1.6.1 Example of Bayesian network

Let us have an example of a simple Bayesian network. We can use the burglar-alarm example from Figure 1.6.
Set of variables is *burglar, earthquake, alarm*. Their data type is Boolean - that means that they are either *true* or *false*.
We use CPT (introduced in Section 1.5) to model conditional probability distribution over the variable given its parents. CPT for this distribution was already shown in Table 1.1.

The advantage of Bayesian network is that we might use variety of different data types for our variables so we are not limited only to `true` or `false`. This is shown in example with paintings [9, p. 141] shown in Figure 1.9 where *categorical* variables are used.

Each variable in this example has its own *domain*. This domain specifies, which values are possible for that variable. We can see that the domain of variable *Subject* consist of two values: People and Landscape. The domain of *Brightness* is {Dark, Bright} and the domain of *Size* is {Small, Medium, Large}. In this example, we can see the conditional probability distribution of each variable given by a table that represents conditional probabilities of children variables given their parent variables. From Figure 1.9, we can see that, for example, the probability that the painting is large given that it is a painting of a landscape is 0.25.

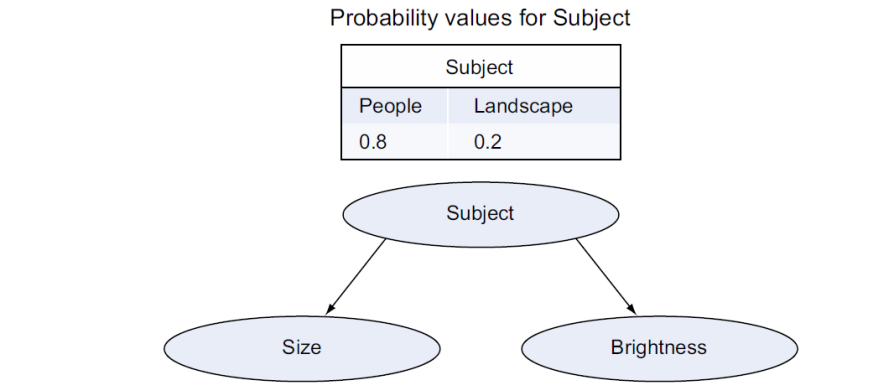



**Figure 1.9:** Example of three-node Bayesian network [9].

# Chapter 2

# Probabilistic Programming

This section is an introduction to Probabilistic Programming (PP) and to a PP language called Figaro [9]. The main part of this thesis - intention estimation of traffic participants - which uses Bayesian network is implemented in Figaro. In this section, we present fundamental information about PP and how to use it.

## 2.1 What is probabilistic programming?

People often need to decide in the situations when they do not directly observe all relevant factors. We can call this decisions *judgement calls*. One of the ways how to decide under uncertainty has been to use a probabilistic reasoning system. This system combines our knowledge with probability laws to determine those unobserved factors critical to the decision in that situation. Probabilistic programming is a new approach that makes probabilistic reasoning systems easier to build. Probabilistic programming is all about providing ways to specify the knowledge and logic to answer questions.

### 2.1.1 Probabilistic reasoning

Probabilistic reasoning is an approach that uses a model of the domain to make decisions under uncertainty. We can show an example on a simple situation from a football match.

Imagine that you have to decide whether following corner kick will result in goal or not. You know that every 10th corner kick results in goal in average. So the probability of scoring from corner kick is 0.1. This is not all you know. You know that the attacking team is great at scoring from corner kicks and their forward, who is $200cm$ tall, is the league's best scorer. Then you know that the goalkeeper of the defending team is one of the shortest goalkeepers in the league, is only $175cm$ tall, and has almost zero experience. This is the knowledge you either found out in some statistic records or observed directly. Besides that, there is a strong wind that makes it difficult to produce long kicks. All these factors are shown in Figure 2.1. How can we figure out the probability that the corner kick will result in a goal?

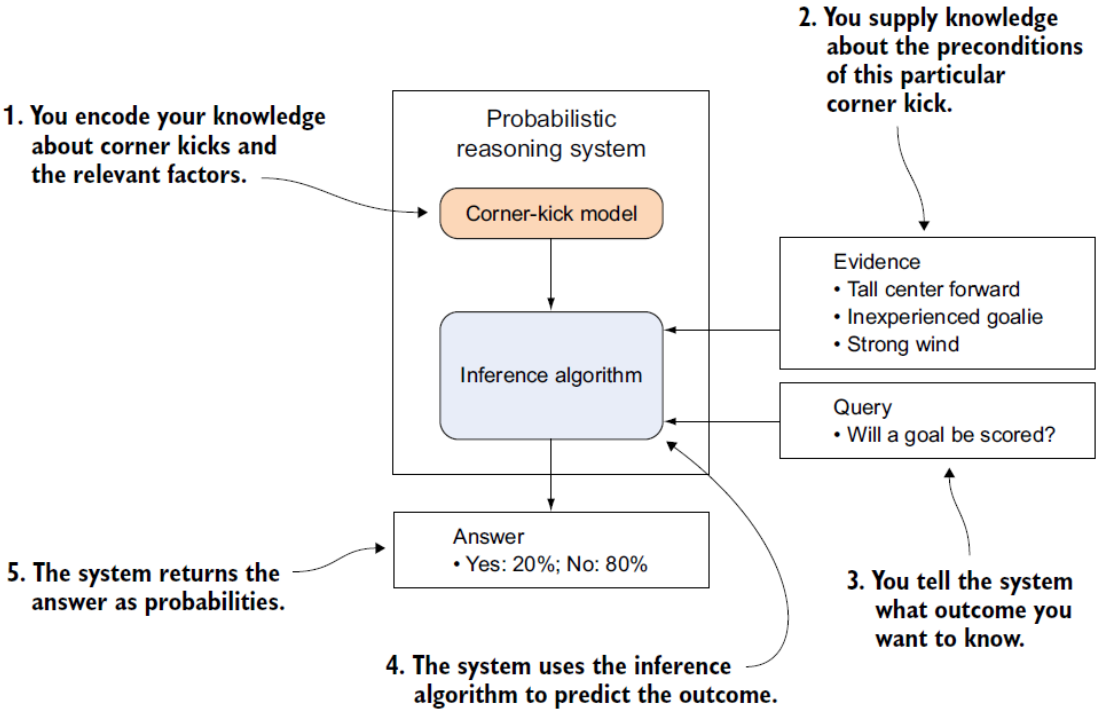



**Figure 2.1:** Model of corner kick decision system [9].

In Figure 2.1 you can see a way how to use a probabilistic reasoning system to find the answer. This has several parts:

1. You need to encode your *knowledge* about corner kicks and all relevant factors in model.
2. You supply the *evidence* you observed - tall forward, short and inexperienced goalkeeper, strong wind...
3. You tell the system you want to know the probability of a goal being scored. In this example the inference algorithm can e.g. return that the attacking team will score a goal with 20% probability.

We can summarize this as follows:

- You create a *model* that captures all the relevant general knowledge of your domain in probabilistic terms.
- In the next step you apply the model to any specific information you have to draw conclusions. We call this specific information *evidence*.
- The relationship between the model, observed evidence, and the answers to queries is well defined mathematically by the laws of probability. The process of using the model to answer queries based on the evidence is called *probabilistic inference*.

We can see all these components in the following Figure 2.2.

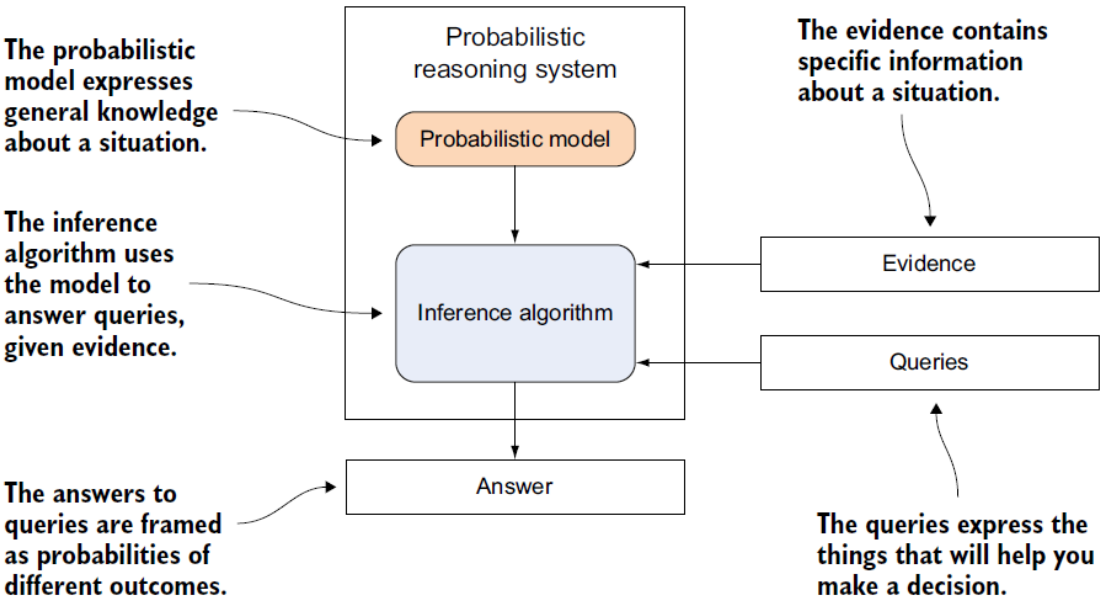



**Figure 2.2:** Components of probabilistic reasoning system in [9].

### Types of probabilistic reasoning system

We can divide probabilistic reasoning systems into 3 types in general:

- *Predict future events.* This is the type of reasoning systems we showed in Figure 2.2. We usually observe some evidence and based on it we can predict future events.
- *Infer the cause of events.* This is basically the reverse case. We observe the outcome of some situation (e.g. a corner kick in football game) and based on this information we infer what caused this result. For example, we can infer whether central defenders of a defending team are good in headers.
- *Learn from past events to better predict future events.* Once we have already observed some evidence in the past, we can use this information to predict more accurately outcomes of the future events.

## 2.2 Figaro

In this thesis, we will use probabilistic programming system called *Figaro*. It is a functional and Turing-complete system. *Functional* means that it is based on rules of functional programming. *Turing complete* is a system that is able to encode any computation that can be done on a digital computer.

*Figaro* itself is implemented as a *Scala* library. *Scala* is a programming language that provides support for functional and object-oriented programming. *Scala* is statically-typed

and type-safe. In Figure 2.3, we can see the way how *Figaro* uses *Scala* to implement a probabilistic programming system.

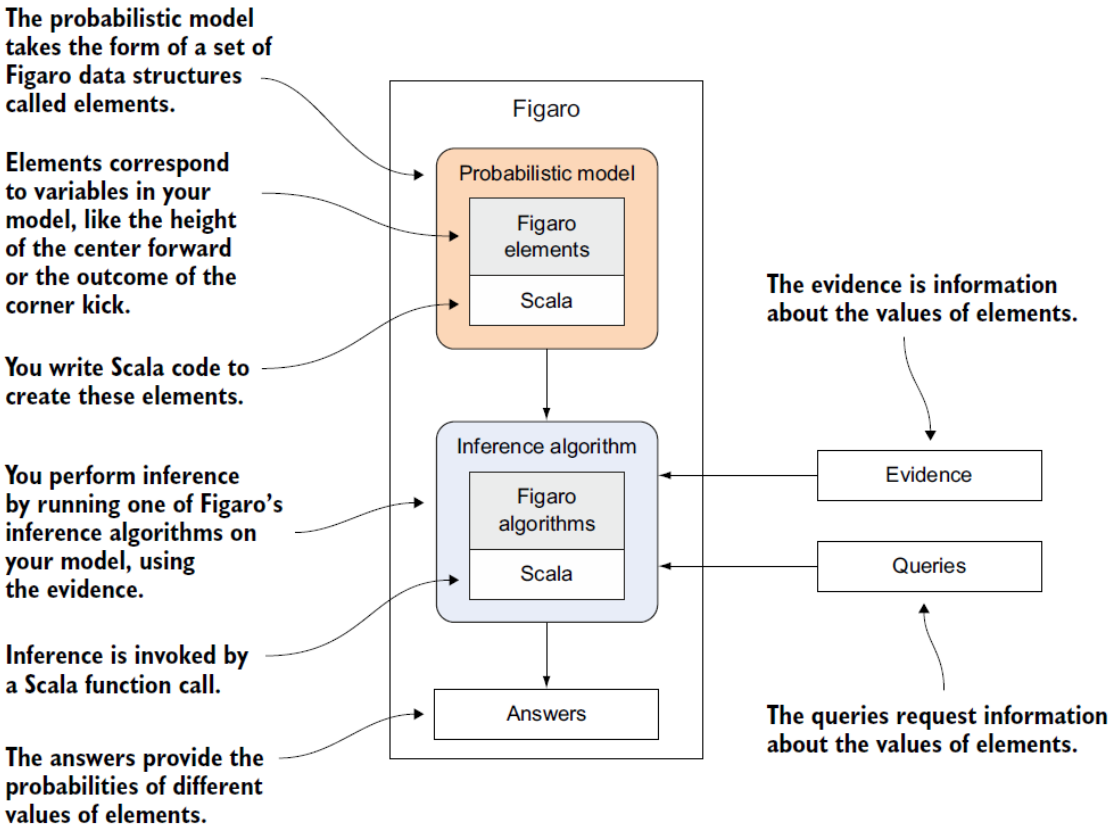



**Figure 2.3:** How *Fagaro* uses *Scala* to implement a PP system [9].

### 2.2.1 Probabilistic model

Figaro probabilistic model consists of a number of data structures called *elements*. Each of these elements represents a variable that can have any number of values. These data structures (elements) are implemented in Scala. We can write a program in *Scala* to create probabilistic model using these data structures. We are able to deliver evidence to the model providing information about the values of elements and we can answer a query about any of them. The information about values might be given in a form of observations, conditions or constraints.

#### Elements

There are two main kinds of element: atomic and compound.

- *Atomic elements* can be viewed as building blocks of the probabilistic model. These elements represent basic probabilistic variables and they do not depend on any other

variables. Atomic elements are further divided to `discrete` atomic elements (such as *Boolean* or *Integer*) and `continuous` elements that are typically represented by *Double*.

- On the contrary, *compound elements* are the connectors. It means that they depend on one or more other elements and they lead to building more-complex elements that are necessary for the model. Figaro provides number of different compound elements. The two most important of them are called *Apply* and *Chain*. These compound elements provide the needed connections between elements. For example, *Chain* takes a probability distribution over a parent variable and a conditional probability distribution (later just 'CPD') to child from parent. The result is a probability distribution over a child.

  We can use CPD tables as well. CPD table is a table that describes conditional probability distribution from the parent to the child by a table. You can see example of one of these tables in Table 1.1. Figaro provides two types of CPD tables. They are called *CPD* and *RichCPD*. When using $CPD$, we need to specify all the possible outcomes. This is not useful for bigger distributions. For these bigger distributions, Figaro provides another constructor called $RichCPD$. In contrast to $CPD$, the case in each clause specifies a set of possible values for each parent, rather than a single value.

### 2.2.2 Inference algorithm

The inference algorithms used to answer queries are implemented in Scala and are invoked by a function call. We simply choose one of Figaro built-in inference algorithms and apply it to our model, to answer the query, given the provided evidence. Results of this inference algorithm are then the probabilities of the query elements.
In Figaro, there are two main types of inference algorithms:

- *Factored algorithms* work by operating on data structures called *factors* that capture the probabilistic model being reasoned about.
- *Sampling algorithms* work by creating examples of possible worlds from the probability distribution and using those examples to answer queries.

#### Factored algorithms

The factor is defined as a representation of a function from the value of a set of variables to a real number.

For example, have a factor over two variables that may have different number of values. The probabilistic distribution assigns a number from interval from 0 to 1 to every possible world that can be created with these two variables. This distribution can be represented by a table conveniently.

In Figaro, there are two main factored algorithms:

- The first one is called *variable elimination algorithm*. This algorithm is an *exact* one. It means that it computes the exact probability of the query, given the evidence. The fact that it is exact might lead to its slow execution.
- Another one is the *belief propagation algorithm*. This one is on the contrary an *approximation algorithm*. It means that it does not computes the exact probability but computes an answer that is most of the time close to the right answer. This leads to the faster performance.

### 2.2.3 Sampling algorithms

As we have already mentioned, sampling algorithms answer queries by generating possible states of variable drawn from the defined probability distribution. Two of the most useful sampling algorithms implemented in Figaro are called *importance sampling* and *Markov chain Monte Carlo*.

Behind sampling algorithms is the *sampling principle*. The basic version of it is very simple. Rather than computing the distribution directly, we can generate a set of *samples*. Each of these samples represents a possible world. The probability of generating each particular world should be equal to the probability of that possible world. After sampling, we can estimate the probability of a possible world by the fraction of samples equal to that world. This is show at 2D example in Figure 2.4.

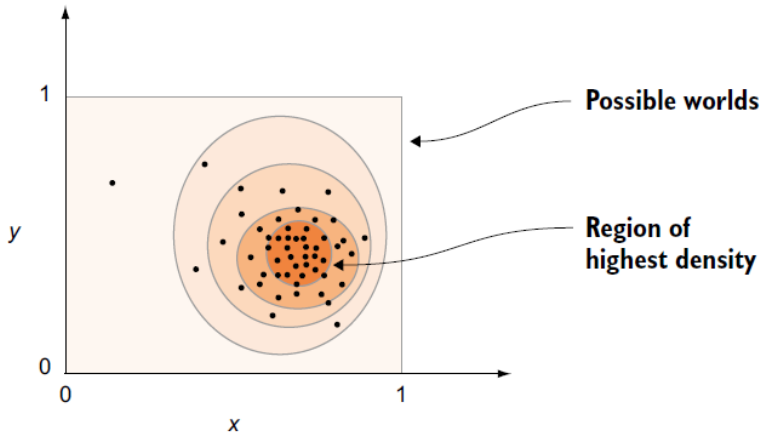



**Figure 2.4:** Example of sampling from [9].

### 2.2.4 Advantages of Figaro

As for the advantages of Figaro, we should mention that it can represent wide range of probabilistic models. We can have any type of Figaro elements. It implements typical types like integers and booleans but we can use arrays, trees or graphs as elements as well. The relationship between these elements is defined by any function. Figaro provides variety of inference algorithms, rich framework for specifying evidence and can represent and reason about dynamic models.

Another great advantage of Figaro is that since it is implemented in Scala, Figaro can be easily used in Java and Scala programs. This is used in this thesis to communicate with the simulator, which is written in Java.

### 2.2.5 Summary

Knowing about the structure of probabilistic models in Figaro, we can summarize what we learned about Figaro in the following Figure 2.5.

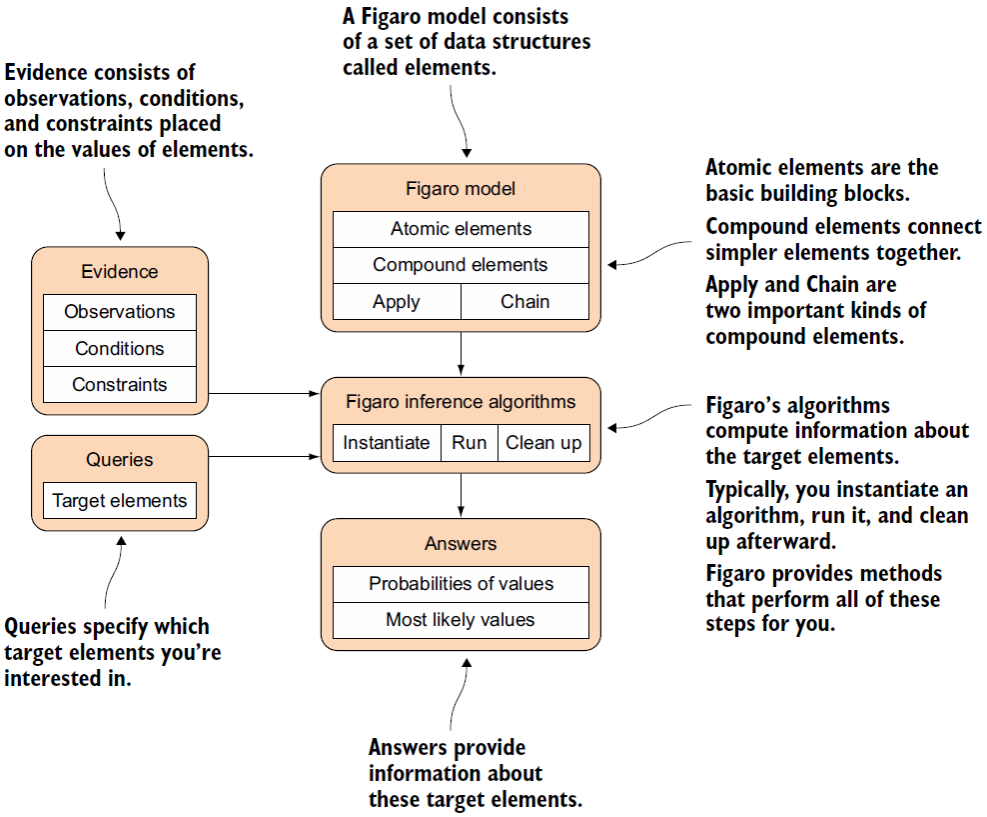



**Figure 2.5:** Summary of Figaro key concepts [9].

# Chapter 3

# Simulator

This part describes the simulator created for testing of an autonomous car model. The simulator is created in Java and provides interface for testing of a car decision system. It contains both visual and text output of ongoing actions. Cars in this simulator should contain features for observation and intention estimation of traffic participants. This part is implemented in Figaro programming language for probabilistic programming and we will describe this part later.

## 3.1 Description of the simulator

The simulator is implemented in Java programming language. It is divided to several classes. In following sections, we will describe each major class along with pointing out some important functions and features.

The simulation itself runs in a cycle. Each turn consists of these steps:

1. Simulation of each cars movement and its behavior.
2. Update and redraw of all components of the simulator (cars positions, traffic light colors, etc.).

### 3.1.1 Entity class

The entity class is an interface for every node / entity used in the simulator map. It prescribes functions for classes that implement Entity interface.

Currently, there are three classes that implement Entity - Crossroad, Crosswalk and ParkPlace. Each of them implements functions from the interface in its own way. Here are few important functions:

- connectCrossroad/Crosswalk - connects a Crossroad/Crosswalk to an Entity

- deleteNeighbour - deletes given neighbour from an Entity
- freeDirections - returns list of unoccupied directions of an Entity
- getNeighbours - returns list of all neighbours connected to an Entity
- getTLColor - if an Entity has a traffic light, it returns current color
- getPedestrians - if an Entity is a Crosswalk, it returns array of all Pedestrians near the Entity; distance that is considered 'near' can be changed easily
- getEnds - returns ArrayList of Entity's Directions
- getPhase - if Entity has traffic lights, getPhase returns current color

### 3.1.2 Crossroad

`Crossroad` is a class that represents node in the map, that stands for a crossroad. It has four directions, in which one can connect other nodes to the `Crossroad`. These directions are named after cardinals ('n' stands for north, 'w' for west, 's' for south and 'e' for east) and are objects of Direction class.

**Crossroad** constructor has only one parameter, a Boolean *trafficLights*, which indicates whether the Crossroad has traffic lights, or not. The traffic lights simulate actual behavior where its color switches from green through yellow and to red. Also, the Crossroad has a variable called *phase* that is updated in each step of the simulation and which determinates color of traffic lights.

Class Crossroad has a few static lists, which groups this type of Entities. These are the lists:

- crossroads - contains all created Crossroads
- crossroadsTL - contains all Crossroads with traffic lights
- crossroadsNoTL - contains all Crossroads without traffic lights

These are other important functions implemented in a Crossroad class:

- getTLColor - this function takes a direction of car approaching the Crossroad as a parameter and returns current color at traffic lights in required direction
- updateTL - updates *phase* variable that determinates the color of traffic lights

### 3.1.3 Crosswalk

Another class that implements Entity interface is `Crosswalk`. This class models crosswalks (zebra crossings) in the simulator.

This class has a static list that contains all Crosswalks called *crosswalks*. This list comes useful in times when we need to update certain variable of each object of `Crosswalk` class.

`Crosswalk` constructor has no parameters. The most important thing about Crosswalk are pedestrians (objects of `Pedestrian` class, Section 3.1.8) in the vicinity of a Crosswalk.

Each pedestrian is described by his current direction of walk, distance left to the crosswalk, its speed and Boolean variable *isCrossing*, which indicates whether the pedestrian intends to cross the road.

We add the pedestrians to the Crosswalk with function *addPedestrians*. This function takes the number of pedestrians as a parameter and creates list of Pedestrians. This list belongs to the `Crosswalk` object and represents pedestrians in the vicinity of zebra crossing.

Another important function is a static function called *update*. This function takes the list *crosswalks* and for each crosswalk in the list takes crosswalk pedestrians and updates their distance from crosswalk. The update is based on their current distance, direction and speed.

### 3.1.4 ParkPlace

Class `ParkPlace` models parking place in our simulator.

This class has a static ArrayList called *parkPlaces* where all instances of this class are kept. Specific variable of this class is Boolean *isOccupied*. Its value shows whether the parking place is occupied by any car at the moment.

### 3.1.5 Car

Class `Car` represents the car in the simulator world that extends Thread class. The behavior of a car is defined in `CarDecisionSystem` class for better clarity. This class will be described separately in Section 3.1.5.

To ensure that no car makes more steps than other car, each car runs its simulation just once in a cycle. The cycle is defined by a variable called *carTurn*, determining which Car should run its simulation currently. The car is able to simulate next step only if its *idNumber* equals *carTurn*.

At the end of every cycle, current position of each car and colors at the traffic lights are drawn into the visualization.

`Car` constructor has several parameters:

- egoCar - is this one car, that observes states of the others?
- wantsToPark - does a car want to park somewhere?

- reach - the maximum distance (in meters), at which the car can observe the state of an Entity, simulates reach of real-world car sensors
- maxSpeed - the maximum speed of a car in meters per second
- acceleration - acceleration of the car in $m/s^2$
- brakes - maximum braking in $m/s^2$
- start - Entity from which the car begins its ride
- finish - the final destination
- path - path of Entities from start to finish; the path is found by Breadth First Search algorithm, that is described in Section 3.1.9.

Other important items and variables of a `Car` class:

- carList - static list of all cars currently driving in the simulator
- lists of observations - lists that keep information about observations of a scene (speed of cars, distances, pedestrians, etc.)
- probsCarAheadMap - this HashMap maps ID number of cars ahead to the observed probabilities of their behavior. These probabilities are obtained from the probabilistic model of the car behavior (described in Section 4)
- distanceLeft - distance left to next Entity
- braking/accelerating - is the car braking/accelerating at the moment? Represented by a boolean variable.
- carAhead - is there a car ahead of the car, that this car needs to consider while deciding about its own actions? Represented by a boolean variable.
- turning - is the car going to change its direction? Represented by a boolean variable.
- indicators - represents, which indicators are on, possible values are: left, right, all or none
- brake lights - brake lights are ON (`true`) or OFF (`false`). Represented by a boolean variable.

Because class `Car` extends class `Thread`, every Car object has its `'string' ID`. Each of this 'string' id consists of 'car' and a unique number. Each car has its color as well. The color is chosen from 6 available colors.

Car class has an enumeration class called `DirectionEnum` with values *left*, *right* and *none*. These values are used for indicating whether the car is giving a sign about changing its direction. It is stored in *directionBlinking* variable.

### Car decision system

The decision system of a car is implemented as an own class. This system is responsible for the behavior of the car. For example, the car decides whether it has to stop at the traffic lights or at the zebra crossing. We will describe several of these functions.

**Function for decisions near the crossroad**
Function for decisions in the near of the crosswalk is called *DecideCrossroad*. This function is called when the car is close to the traffic lights. It decides based on the current color at the traffic lights. The decision system can be show in Table 3.1

| Color | P(braking) |
|---|---|
| red | 1.0 |
| green | $0.1 \pm 0.05$ |
| yellow | $0.5 \pm 0.1$ |

**Table 3.1:** Deciding at the traffic lights

where *Color* is the current color of the traffic lights and *P(braking)* is the probability that the car will brake.

**Function for simulation**
*The most important function* of Car decision system class is a function used for simulation called `simulateStep`. This function simulates single step of the car in every cycle. It is called in a loop inside the `run()` function.

This function takes a sequence of Entities from start to finish as a parameter. The sequence is obtained by Breadth First Search (BFS) algorithm which will be described in Section 3.1.9.

First, the function sets the Boolean variables called *braking*, *carAhead*, *accelerating*, *overtaking* and *turning* to `false`. It means that the car has no reason for braking now, there is no car ahead that might cause braking currently, the car is not overtaking any other car, and is neither accelerating or changing its direction.

Second, the function checks the remaining distance to the next Entity.

- If the remaining distance is *less than, or equal to* specified distance (this distance can be easily modified) the car decides whether it should blink its indicator. This is done inside the `setBlinking` function. If the car should use indicators it does so with probability of 0.9.
  In case that there is a parking place ahead, the car is less than 100 meters far and wants to park there, it flashes right with probability of 0.8 and does not flash otherwise.

- If the distance is *less than or equal to zero* and the car has not reached the finish yet, it sets a new goal/direction by picking next Entity from the sequence of Entities passed to the function as a parameter '*path*'.

Next the car observes its surrounding. A function called `observeScene` first checks whether there is any Entity (crossroad, crosswalk or park place) in the reach. If so, it obtains evidence about state of an Entity and decides whether the car should brake based on the given evidence. Next the `observeScene` function checks whether there are any cars ahead and decides whether the car should break inside the `checkRoad` function.

**checkRoad function**
`checkRoad` function first checks whether there is any car ahead in a reach of sensors. If so, it checks the speed of the car ahead. If the car ahead is faster, then there is no reason for more action. If the speed is less than or equal to the car speed, this function checks whether the car can overtake the car ahead.

This is done by checking whether there are any cars driving in an opposite direction that would make the overtake. Another condition is that the overtake would be completed before the car reaches next crossroad or crosswalk (since it is forbidden to overtake there). This is done inside `canOvertake` function. If the car can overtake the car ahead, it does so. Otherwise the car starts to brake. In case the car is able to brake to speed of the car ahead in just one step, it does so.

Finally, the `simulateStep` function either brakes or accelerates the car based on previous decisions. Next it sets the distance left to the next Entity. It is computed from the distance previously left and from the car's current speed.

At the end, the function computes a new position of the car in the map, draws the car to the map visualization, sets whether the brake lights are on or off and sets the current state of the car.

### 3.1.6 Direction

`Direction` class represents the direction, in which one Entity can connect to another. It can be viewed as a road connecting two entities as well.

This class has these important variables:

- direction - character representing one of cardinal points ('n' for north, etc.)
- hasNeighbour - Boolean value which indicates whether in this direction there is any other Entity connected
- neighbour - Entity of a neighbour connected in this direction
- distance - distance in meters to the neighbour
- cars - list of Cars driving in this Direction (driving certain road)

### 3.1.7 Map

Class Map provides the visualization of the simulator.

The class consist of these parts:

- the map itself that is stored in a 2D array
- frame - JFrame used for visualization of a map
- status - JTextArea used as a status bar with current information about the car

The class provides functions for painting components of a map, which are called in the end of each step of a simulator.

### 3.1.8 Pedestrian

Pedestrian class is a model of pedestrian near the Crosswalk.

Each pedestrian has these variables:

- currentDirection - direction, in which the Pedestrian is currently walking, it has four possible values according to the cardinal points ('n' for north, etc.)
- approachDirection - direction, in which the Pedestrian is approaching the Crosswalk
- distance - distance to the crossroad in meters
- speed - speed of the pedestrian in meters per second

The most important function is *nextDirection()*. This function first checks the distance of a pedestrian from the crosswalk. If the distance is less than or equal to predefined value (in meters) it decides whether the pedestrian keeps walking straight on (with probability $0.6 \pm 0.1$) or whether decides to cross the street.

The probability of crossing the street depends on pedestrian current direction. In the case that pedestrian walks towards the street, he crosses it with $0.8 \pm 0.1$ probability, otherwise (he is walking straight on) pedestrian crosses with probability of 0.1.

### 3.1.9 Algorithms

Algorithms class currently consists of several algorithms and some support functions.

Among these support functions, belongs a useful static method called `opposite`. This function takes a direction and a crossroad as parameters and returns the opposite direction to the given direction at the given crossroad. This is useful for checking the situation on the road ahead.

Another function is `newMapPositions`. This function takes parent and child Entity, the direction between them and the direction of connection from parent to child. It sets a position in the map to the child Entity.

Function `oppositeChar` returns a one-letter shortcut of an opposite direction to the given direction. For example, for given character '*n*' (north) it returns '*s*' (south).

### Breadth First Search

The Breadth First Search (BFS) algorithm is used for graph exploration and for route planning. The algorithm returns a sequence of Entities from the start to the finish.

This class has a private inner class called `ExtendedEntity` that extends `Entity` by its parent and by the depth in a graph, in which was the `Entity` explored. This inner class is used in `BFS` algorithm.

BFS algorithm is used because of its easy implementation. It also finds the path that leads through least number of nodes (in our case entities). As the entities are not necessarily equally distant from each other, it might not find the shortest path. This is not a problem in the application of this simulator. We do not seek the simulator where the cars take the shortest possible path from start to finish. We need this simulator primarily for testing outputs of Bayesian Network capable of predicting the intentions of the traffic participants.

### 3.1.10 Run

`Run` class contains `main` method that is used for starting the whole simulator. We can see the visualization in Figure 3.1. The blue squares are Crosswalks and the green squares are Crossroards which might or might not have traffic lights.

Main body of this class creates all Crosswalks and Crossroads first, then connects them together, adds `Pedestrian`s to the `Crosswalk`s and finally creats Cars. Next it starts a thread for each Car. This begins the car movement and simulation inside a loop. Inside this loop, one can choose how long does each step takes by setting Thread.sleep(`time`).

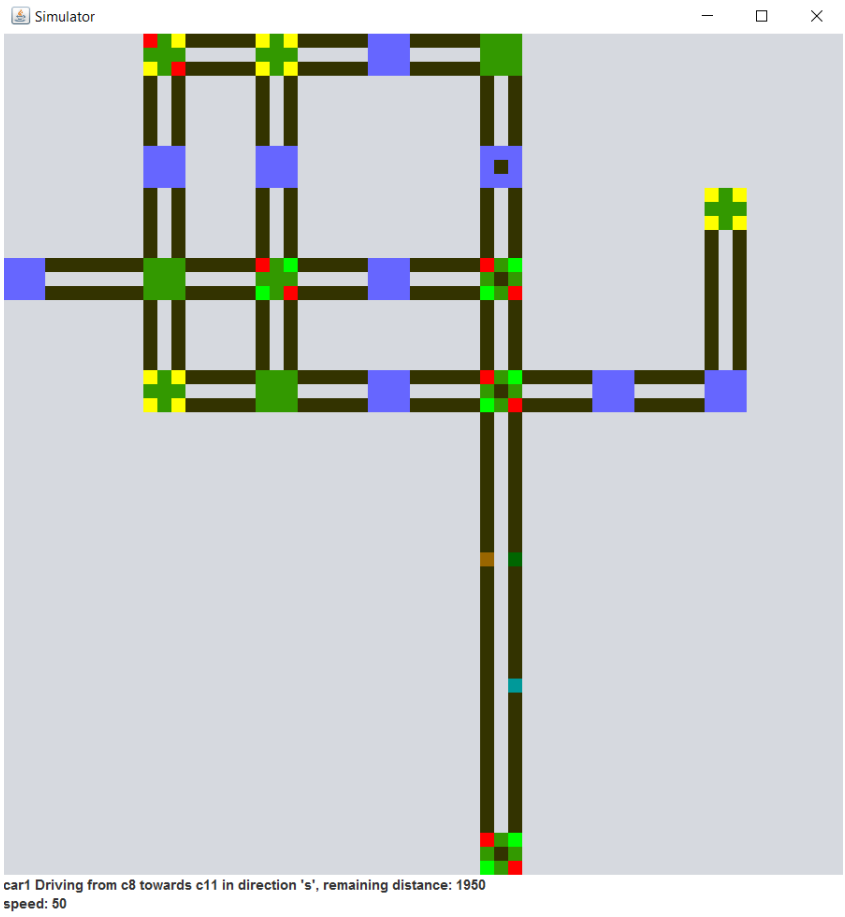



**Figure 3.1:** Screenshot of a simulator visualization.

# Chapter 4

# Bayesian model for Simulator

This part describes a Bayesian network and its usage with data obtained from the simulator. When the developement of Bayesian network started, we did not have any data measured in real-life situations. The simulator allowed us to familiarize with Bayesian networks and try some approaches for given problem.

This Bayesian network uses several objects from the simulator. It observes other cars in the reach of sensors, pedestrians near the crosswalk, tracks distances left to junctions and much more.

We begin the description with the whole model of Bayesian network, which used the data from simulator. Next, we describe vital parts of the model. We also describe the way we learned the parameters of the network.

It is important to note that all information, that inputs Bayesian network, is observed by one specific car. In this thesis we call this car *EGO car*. The values observed about cars are not about this *EGO car* but represent state of other cars observed by *EGO car*.

## 4.1 Bayesian network

The Bayesian network used in this model is shown in Figure 4.1. The network takes into account information about pedestrians, zebra crossings, cars, traffic lights and junctions. Figure 4.1 is shows that we have three major outputs about car behavior: *braking*, *stopping* and *turning*. We have focused to the *'braking'* part so far. The other two outputs are implemented in the network as well, but parameters of their distributions are not yet properly learnt and will be completed in later stages of project.

### 4.1.1 Measured values from simulator

We will describe measured values that we can use as inputs of the network.

**pedestrian**:

- origin - origin of the pedestrian's approach to zebra, its value is restricted to cardinal points (North, East, South, West)

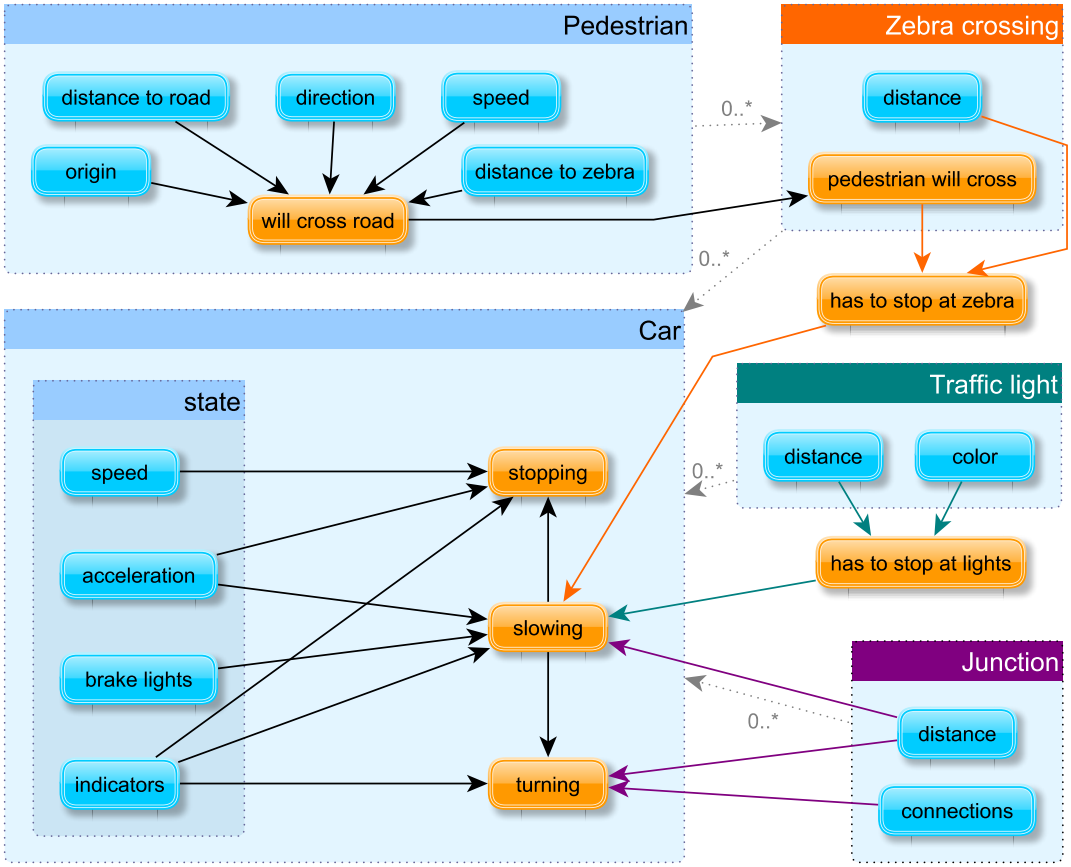



**Figure 4.1:** Model of Bayesian network used in simulator.

- distance to road - pedestrian's distance to the road
- direction - current direction of pedestrian's movement, its value is restricted to cardinal points
- speed - speed of pedestrian
- distance to zebra - pedestrian's distance to the zebra

**zebra crossing**:

- distance - car distance to the zebra crossing

**car**:

- speed - speed of the car
- acceleration - acceleration of the car
- brake lights - indicates whether the car is braking

- indicators - indicates which indicators are on, possible values are: *none, left, right, all*

**traffic lights**:

- distance - car distance to the traffic lights
- color - current color at the traffic lights

**junction**:

- distance - car distance to the junction
- connections - possible directions that the car can take from this junction

It is important to note that we do not use all of these measurements in our Bayesian network since we do not have all of these values validly measured in data obtained from real-life situations.

## 4.2 Distributions

We describe important distributions used. We take each part of the network alone. Most of the probabilities are represented as learnt parameters and are not hard-coded. We describe the learning process in Section 4.3.

### 4.2.1 Pedestrian

As we have already written in Section 4.1.1, the inputs are: *origin, distance to road, direction, speed* and *distance to zebra*.

Output of this distribution is the probability that the pedestrian will cross the road. When there are more pedestrians near the zebra crossing, we take only the highest probability into account.

Before we get to the Pedestrian distribution itself, we need to define one more distribution. It indicates how close the pedestrian to the zebra crossing is. This distribution has a single input - *pedestrian's distance to zebra*. If the pedestrian is closer to the zebra than the specified value (in metres), the distribution has likelihood 1.0. Otherwise the likelihood decreases exponentially. The whole distribution is shown in Figure 4.2. We will use this distribution few more times in the following section of the thesis.

This probability distribution of a pedestrian crossing the road is specified by a conditional probability distribution (CPD) table. In Figaro, there are two possible ways to use CPDs as we have already mentioned in Section 2.2.1. For this distribution, we will use *RichCPD* with advantage (Table 4.1).

The output variables *straightCrossing* and *towardsCrossing* are learned parameters. Value of *straightCrossing* is the probability that the pedestrian wants to cross the street

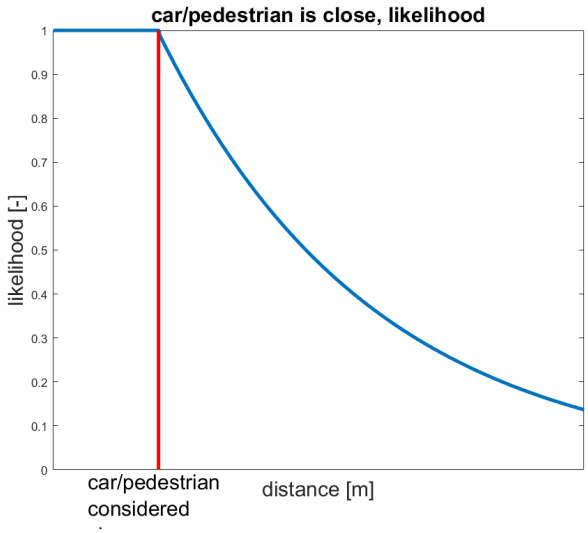



**Figure 4.2:** Distribution of likelihood for pedestrian.

when is in the vicinity of the zebra crossing and is walking in the direction parallel to the road. Value of *towardsCrossing* is the probability that the pedestrian wants to cross when walking towards the road.

| approach direction | current direction | pedestrian close | P [-] |
|---|---|---|---|
| North | South | true | straightCrossing |
| East | West | true | straightCrossing |
| North | West | true | straightCrossing |
| West | East | true | straightCrossing |
| North | West | true | towardsCrossing |
| East | North | true | towardsCrossing |
| South | East | true | towardsCrossing |
| West | South | true | towardsCrossing |
| * | * | * | 0.0 |

**Table 4.1:** CPD table for probability of pedestrian crossing the road

The symbol $*$ means that the parent can take any value and the clause will apply, assuming the other parents have appropriate values.

### 4.2.2 Zebra crossing

This distribution models how likely it is that the car has to stop because there is a pedestrian likely to cross the road. It has two inputs

- distribution function for *distance* of EGO car to the zebra crossing and
- the *highest probability* of a pedestrian crossing the road.

The distribution function for EGO car *distance* to the zebra crossing represents how close is the car to the zebra crossing. It has a shape similar to the distribution of pedestrian probability to cross (Figure 4.2). The parameters of this distribution are different, of course.

### 4.2.3 Traffic lights

This distribution models the probability that the car has to stop at the traffic lights. It has two inputs:

- distribution of *car distance* to the traffic lights and
- *current color* at the traffic lights.

Distribution of *car distance* indicates car closeness to the traffic lights. Shape of this distribution is similar to the distribution shown in the Figure 4.2, but with different parameters. *Current color* can have values `red`, `yellow` or `green`. These two inputs are chained together by CPD Table 4.2.

| car is close | current color | probability |
|---|---|---|
| true | green | greenClose |
| false | green | greenFar |
| true | yellow | yellowClose |
| false | yellow | yellowFar |
| true | red | redClose |
| false | red | redFar |

**Table 4.2:** CPD table for car's distance.

Learned parameters that represent probabilities have the name corresponding to their meaning. For example, *greenClose* is probability, that the car has to stop if it is close and there is a `green` color at the traffic lights.

### 4.2.4 Car slowing

Distribution **Car slowing** represents probability that the observed car in front of a EGO car will cause EGO car to slow down in the next two time steps. It has a lot of inputs:

- *acceleration* of car ahead
- *brake lights* of car ahead
- *indicators* that have the car ahead turned on
- *likelihood of stopping because of pedestrian crossing the road* (Section 4.2.2)

- *likelihood of stopping because of traffic lights* (Section 4.2.3)
- *distance* to the junction ahead

The **Car slowing** distribution is created by a combination of `If` compound elements and `CPD` tables. Some of the inputs are an input of some other distribution. Now, we will describe these distributions.

**Car is braking** (*isBraking*):
*isBraking* is `true` in the case that the car is braking (that is decided from the value of acceleration). Otherwise its value `false`.

**Entity in reach** (*entityInReach*):
It is `true` if the entity (zebra crossing, junction, etc.) is in the reach of car sensors. If it is out of the sensor reach, its value is `false`.

**Car ahead in reach** (*carAheadInReach*):
Similar to the *entityInReach*. It is `true` if there is any car ahead in the reach of sensors and `false` otherwise.

**Car is close to the entity** (*carCloseEntity*):
This distribution was already described in Section 4.2.3. It represents how close is the car to the zebra crossing, crossroad or to the park place. Its shape is similar to the distribution in Figure 4.2, just with different parameters.

The model of this **Car slowing** distribution is shown in Figure 4.3.

The only hard-coded probability is a probability of 1.0 in the case that we know that the car is already braking at the moment.

In the case that there is no entity (junction, zebra crossing,...) in the reach of sensors and there is any car ahead in the reach of sensors, the probability of braking is equal to the learned value *notReachCar*. If there is no car ahead, value of this probability is *notReachNoCar*.

If there is an entity in reach, it depends on whether there is any car ahead and whether the car is close to the entity. These two values decide, which CPD table we choose for decision. There are four of these CPD tables and each of them has two inputs:

- probability, that the car needs to stop, because of the traffic lights (*trafficLightStop*) and
- the maximum probability of pedestrian crossing the road (*MaxProbPedestrian*).

We show only one of these CPD tables since the other three are almost similar. CPD table for the car ahead in reach and entity close is shown in Table 4.3.

Output values called *reachCloseBrakeTLCar* and *reachCloseBrakeTLCar* are learned values. The value *reachCloseBrakeTLCar* represents the probability that the car has to stop in the case that it is:

- in the reach of an entity,
- there is a car ahead in the reach of sensors,
- the car is close to an entity and
- it has to stop because of the traffic lights and

Value *reachCloseNotBrakeTLCar* represent almost the same probability. The only difference is that the car does not have to stop because of the traffic lights.

Unfortunately, we have to note that the learned parameters currently do not take into account the maximum likelihood that a pedestrian wants to cross the road. This will be fixed in future work.

| trafficLightStop | maxProbPedestrian | P[-] |
|---|---|---|
| true | true | reachCloseBrakeTLCar |
| true | false | reachCloseBrakeTLCar |
| false | true | reachCloseNotBrakeTLCar |
| false | false | reachCloseNotBrakeTLCar |

**Table 4.3:** CPD table for car ahead in reach and entity close

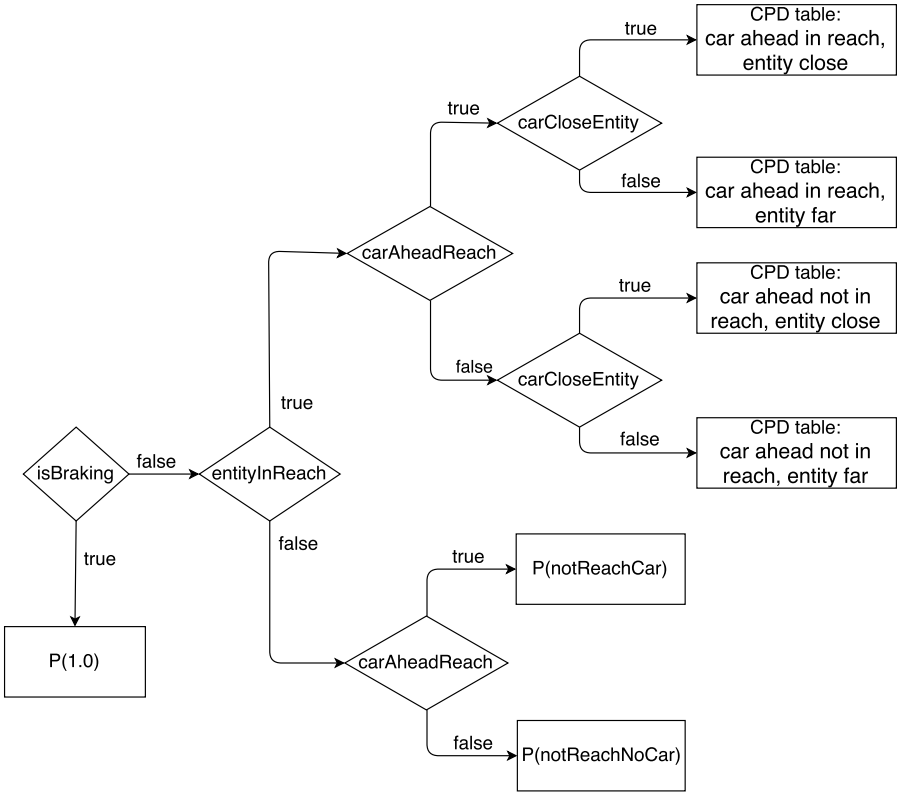



**Figure 4.3:** Diagram of Car slowing distribution.

## 4.3 Learning parameters

The parameters are learned by storing the relevant values to the distribution in the current time step. The values are saved to the outcome. The outcome is not saved just from current time step but also from the next time step, which allows to predict to the future. We will describe our approach for learning the parameters.

### 4.3.1 Parameters for pedestrian crossing the road

The decision about whether the pedestrian is going to cross the road, or not, depends on pedestrian's distance to the zebra crossing and on its movement direction. We recognize two possible directions of pedestrian's movement. Pedestrian can either walk parallel to the road or walk towards the zebra crossing.

We save pedestrian's distance and direction and whether the pedestrian did cross the road in that situation. We save these values in lists. Once the simulation is finished, we pass these lists to function called `processNewObservationsPedestrian`, which processes new observations about pedestrian behavior. Processing of these observations is shown in Figure 4.4.

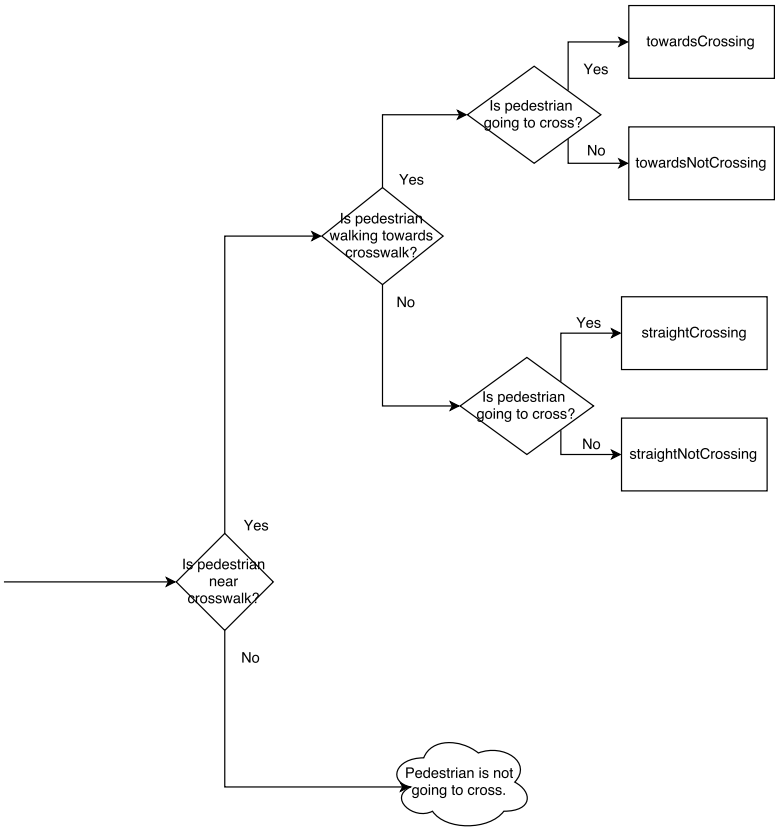



**Figure 4.4:** Function for processing new observations about pedestrians.

As you can see in Figure 4.4, we have four different situations.

- pedestrian is walking *towards* the road and *wants to cross*
- pedestrian is walking *towards* the road and *does not want to cross*
- pedestrian is walking *along* the road (straight on) and *wants to cross*
- pedestrian is walking *along* the road and *does not want to cross*

For example, value called *towardsCrossing* means that the pedestrian is walking towards the zebra crossing and wants to cross the road. Now we need to model the probability that the pedestrian will cross the road when she is near to the zebra crossing. We will decide based on the direction of pedestrian movement. For this, we will use Beta distribution together with Bernoulli distribution.

**Beta distribution**
Beta distribution is a continuous probability distribution defined on interval $[0, 1]$. It is parametrized by two positive parameters called $\alpha$ and $\beta$. Beta distribution is the conjugate prior probability distribution for the Bernoulli distribution. We will use this in a short time. In our case, parameter $\alpha$ is number of times when the pedestrian did cross the road in a considered situation (we denote them with ending `A`), $\beta$ is number of times pedestrian did not cross (denoted with ending `B`). For example, Beta distribution used to model probability that the pedestrian will cross the road when walking towards the zebra crossing (we call it '*towardsCrossing*') is computed as follows (Equation 4.1):

$$towardsCrossing = \text{Beta}\left(towardsCrossingA, towardsCrossingB\right) \tag{4.1}$$

These Beta distributions are then passed to the `Flip` function in Figaro. This function models *Bernoulli distribution*. Using Beta distribution that is the conjugate prior of the Bernoulli distribution is a good way how to represent our prior knowledge about situations.

### 4.3.2 Parameters for braking at the traffic lights

The decision whether the car has to stop or not is based on two values - the car distance to the traffic lights and current color of the traffic lights.

These two values are saved in the every time step to a List. Along with distance and color, we also need to save whether the car needed to brake in the situation, or not. For this, we use a function called `needToStop`.

**`needToStop`** The needToStop function takes into account the car speed and the distance to traffic lights and *phase* of the traffic lights (the *phase* was described in Section 3.1.2). The function works as follows:

1. first computes the *phase* of traffic lights, in which the car will be going through the junction.
2. Based on the *phase*, it finds out the color of traffic lights at that given *phase*.

3. If the future color of traffic lights is *red* the function returns `true` (the car needs to stop) otherwise the function returns `false`.

All of those saved values are processed at the end of the simulation. Values are passed to the function called `processNewObservationsTL`, which processes new observations about car behavior in the vicinity of the traffic lights. This function separates observed situations into a few cases. These cases are named after the situation they describe. For example, case named *greenCloseAndStop* stands for the situation when there was a green color at traffic lights, the car was close and it needed to stop. We get 12 different cases in total. Diagram of this function is shown in the Figure 4.5.

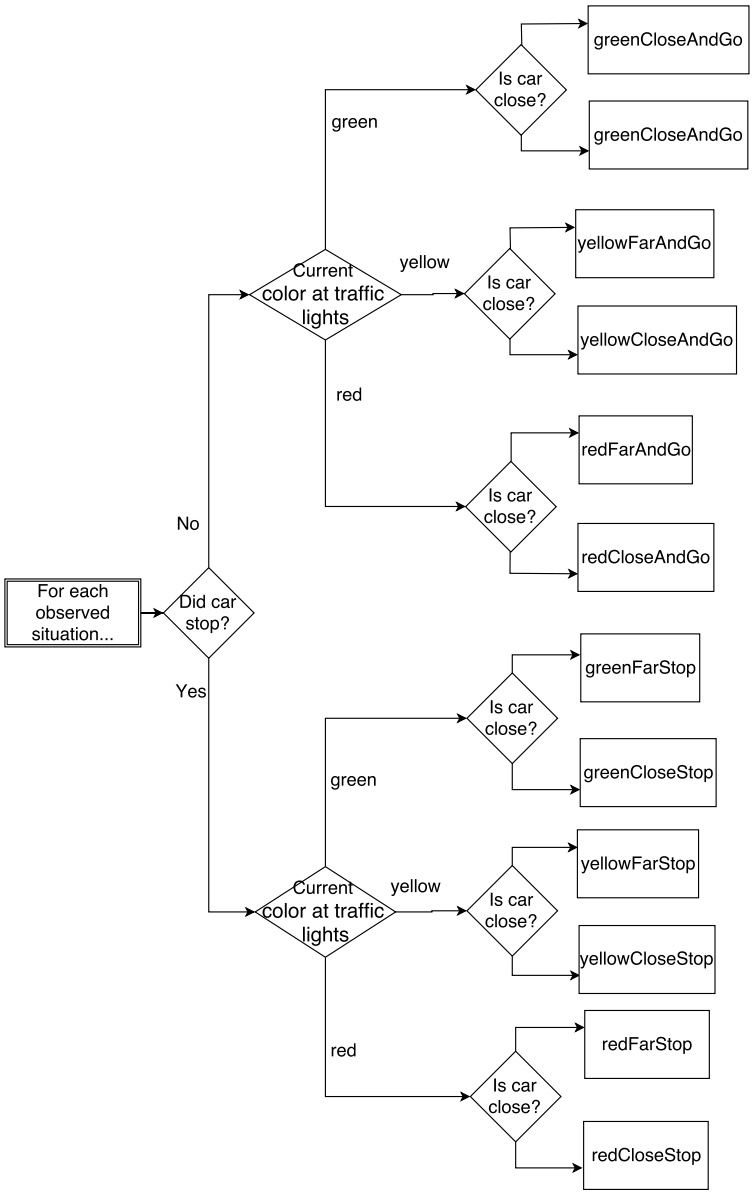



**Figure 4.5:** Diagram for decisions at the junctions (processNewObservationsTL function).

After the function finishes, we have the number of occurrences of each case. We could

have for example measured occurences, 10× *greenFarStop*, 5× *yellowCloseAndGo* and 6× *redCloseStop*. We add this measurements to already measured values stored in a `JSON` file. This allows us to obtain measurements in a number of simulations. Situations, when the car brakes, are stored with ending `A` and situations, when the car does not brake, are stored with ending `B`. The `A/B` ending refers to the $\alpha$ and $\beta$ parameters of Beta distribution.

These measurements define the Beta distribution of each case. Suppose that we want to model the probability, that the car has to stop, when there is red at traffic lights and the car is far. Let us call this *redFar*. The Beta distribution with measured values looks as follows:

$$redFar = \text{Beta}(greenCloseA, greenCloseB). \tag{4.2}$$

The use of too much measurements can lead to overfitting, which is unwanted. Therefore, we have tried to avoid overfitting by choosing reasonable number of measurements. We have not used any sophisticated method for avoiding overfitting and chose the number of measurements experimentally.

### 4.3.3 Parameters for car slowing

These are the parameters used in distribution '*Car slowing*' (Section 4.2.4). The learning process is similar to already described processes in Sections 4.3.1 and 4.3.2. The diagram of processing observed situations is shown in Figure 4.6. *'Should car brake?'* means whether the car should stop because of the traffic lights and *'Will car brake?'* means whether the car did actually brake in the next two steps.

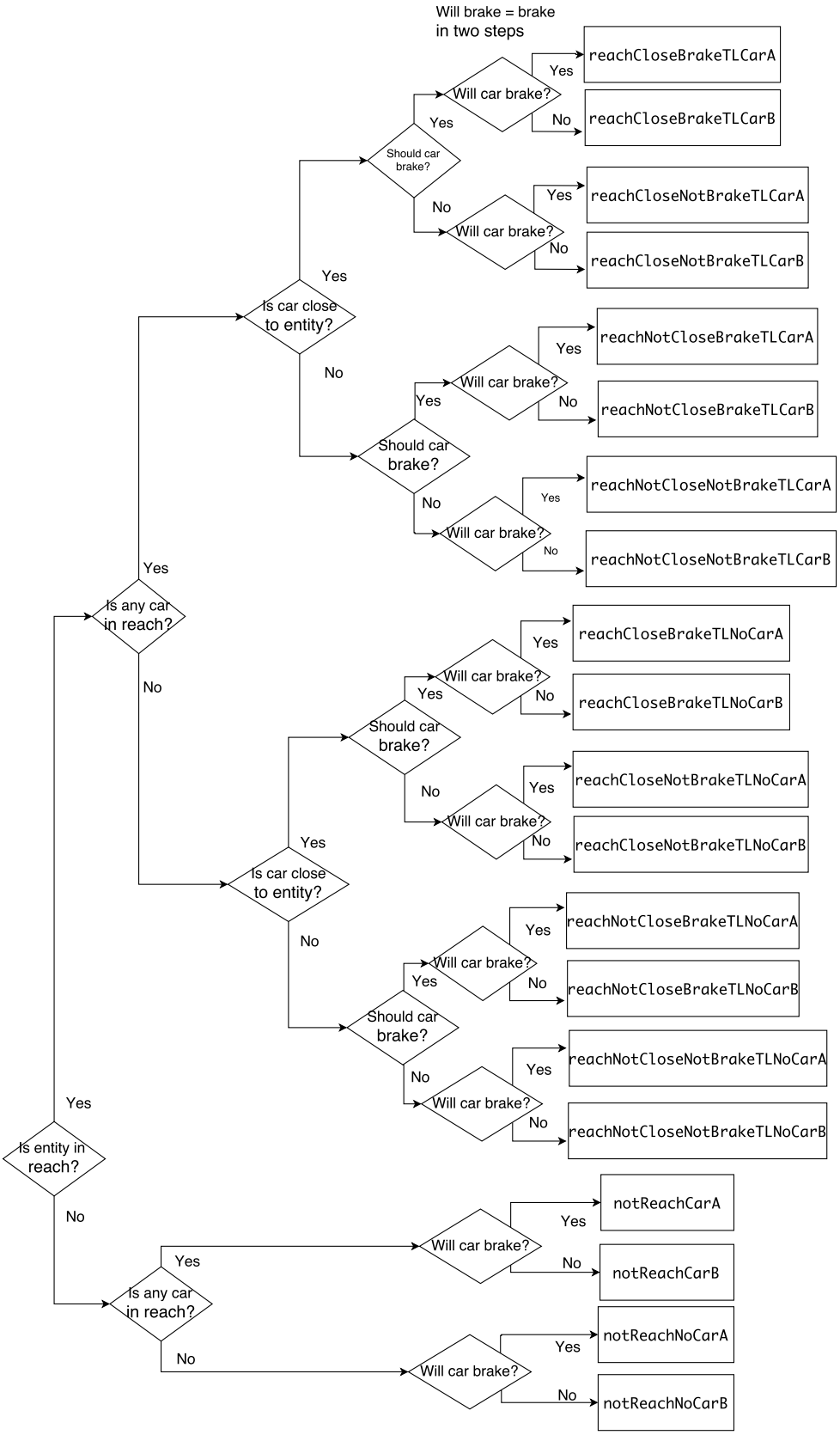



**Figure 4.6:** Learning parameters for car slowing distribution.

# Chapter 5

# Pedestrian's intention estimation near the zebra crossing

This section describes the Bayesian model used when estimating the pedestrian's intentions near the zebra crossing. This bachelor thesis is motivated by the industry - namely by H2020 European project UP-Drive (Automated Urban Parking and Driving). It is worth noting, that this model is used with real-measured data and was not one of the goals at the beginning of this thesis.

We start by describing the model of Bayesian network, which is used for intention estimation of pedestrians. Consequently, we provide the details about the implementation of vital parts of the network and the processing of real-measured data. We also show the visualization of real-measured data. The shown distributions are only illustrating and do not reflect real parameters of used distributions.

## 5.1 Bayesian network

The Bayesian network used in this model is shown in Figure 5.1. We can notice that this network does not take into account only current measured state but previous states as well. This helps the network to provide us better results than if we only used network that would decide about current situation based on data measured only in current time step.

### 5.1.1 Inputs of model

We will describe each input of the model. The distances are measured in metres and angles are measured in radians with respect to the axis aligned with the circle of latitude.

- *Perpendicular distance to zebra crossing* (later as *pedDistanceZebra*) - distance of the pedestrian from zebra in the direction parallel to the road. It can have only values from $\mathbb{R}_0^+$, since we measure the distance from the middle of the zebra crossing.

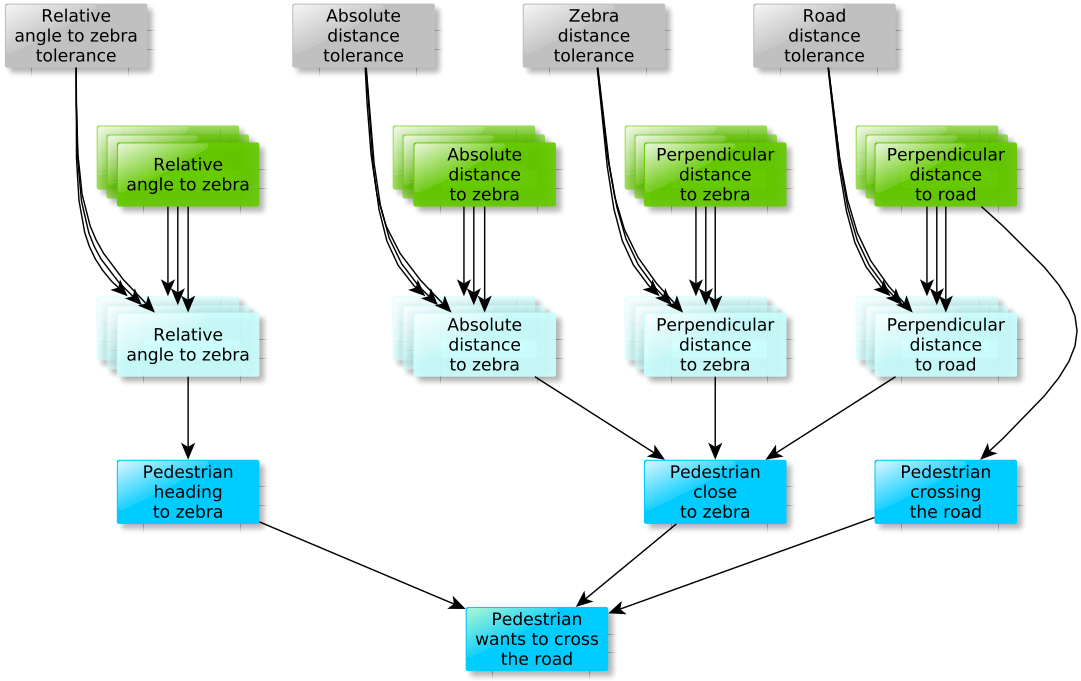



**Figure 5.1:** Bayesian network for the pedestrian intention estimation.

- *Perpendicular distance to road* (later as *pedDistanceRoad*) - distance of the pedestrian from the edge of the road. The value can be both positive and negative. Negative values mean that the pedestrian is walking on the road and therefore is most probably crossing the street.
- *Absolute distance to zebra crossing* (later as *pedDistanceTotal*) - distance of the pedestrian to the edge of crosswalk, it takes into account values *pedDistanceZebra* and *pedDistanceRoad*. It can only have values from $\mathbb{R}_0^+$. The visualisation of these distances is shown in Figure 5.2.
- *Relative angle to zebra crossing* (later as *pedAngle*) - value of this angle is the difference between current direction (vector) of pedestrian's walk and the direction of the zebra crossing as is illustrated in Figure 5.3. Value *pedAngle* can have values only in range $\langle 0, \pi \rangle$ in radians, As it can be seen in Figure 5.3. If we would measure the difference of these directions just clockwise or counterclockwise, it would be possible to measure even greater differences (in $\langle 0, 2\pi \rangle$). That is why we choose the smaller value of difference in clockwise or counterclockwise and take its absolute value.

### 5.1.2 Probability distributions

Now we describe each used distribution function. We have four *main* types of distributions:

- distribution function for pedestrians closeness to the zebra crossing,

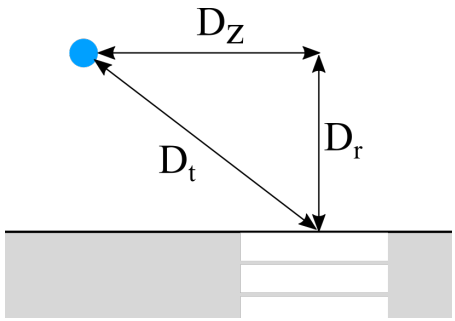



**Figure 5.2:** Demonstration of the distance computation.

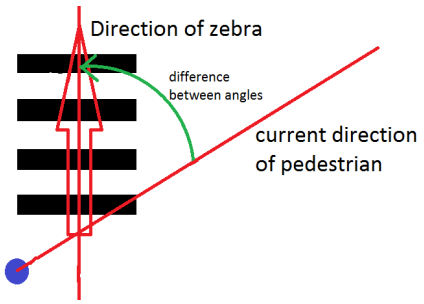



**Figure 5.3:** Computation of pedAngle.

- distribution function for pedestrians angle to the zebra crossing,
- distribution function indicating whether pedestrian is crossing the street at the moment,
- distribution function indicating the probability that the pedestrian wants to cross the street.

### Distribution for pedestrians closeness to the zebra crossing in parallel direction to the road

This distribution is used to indicate distance of pedestrian to the zebra crossing in the direction parallel to the road. It is related to how likely is the pedestrian to cross the street (later denoted as *pedClose*).
As an input, there is an information, which indicates whether the pedestrian is near the crosswalk in the current/previous time step.
This output distribution indicates, how likely is the pedestrian to cross the street based in input information. This distribution is specified by the conditional probability table, see Table 5.1.

**Inputs** of CPT in Table 5.1 are

- current/previous closeness to the zebra crossing - distributions indicating whether the total distance of pedestrian to the zebra crossing is considered close.

| current | previous | Probability |
|---|---|---|
| true | true | 1.0 |
| true | false | 0.7 |
| false | true | 0.4 |
| false | false | 0.1 |

**Table 5.1:** CPT for pedestrian crossing the street.

The input distribution indicates how close the pedestrian is to the zebra in the direction parallel to the street. If the pedestrian is closer to the zebra than the specified value, the distribution has the likelihood 1.0. If the pedestrian is farther than this value, the likelihood decreases exponentially. The distribution is shown in Figure 5.4.

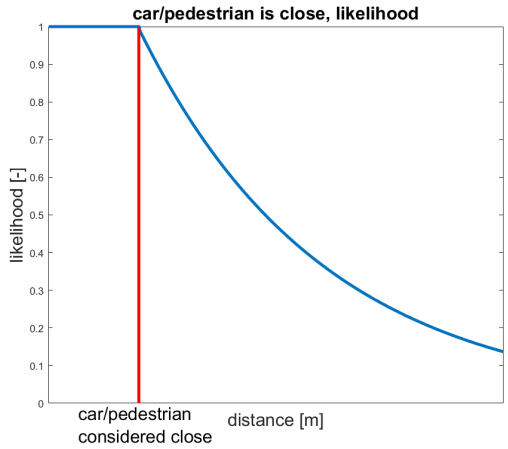



**Figure 5.4:** Distribution for car/pedestrian closeness.

The distribution for the total pedestrian distance (denoted later as *pedCloseTotal*) and for the pedestrian distance to the road (denoted as *pedCloseRoad*) are designed in the same way - we use current and previous values that are inputs of Table 5.1. The current and previous closeness of pedestrian in total is similar to the distribution shown in Figure 5.4. Pedestrian's current and previous closeness to the road is designed in a slightly different and will be described next.

### Distribution for current and previous pedestrians closeness to the road

**Inputs:**

- *Current* pedestrian closeness - indicates distance of the pedestrian in the current time step,
- *Previous* pedestrian closeness - indicates distance to the road in the previous time step.

This distribution is a little bit different than the previos one. The difference is that in the distribution of **pedestrian closeness to the road** it is possible that its input value is negative. This leads to the fact, that this distribution has likelihood 1.0 even for the negative values of the pedestrian distance to the road. The negative distance to the road means that the pedestrian is crossing the street at the moment. The distribution is shown in Figure 5.5.

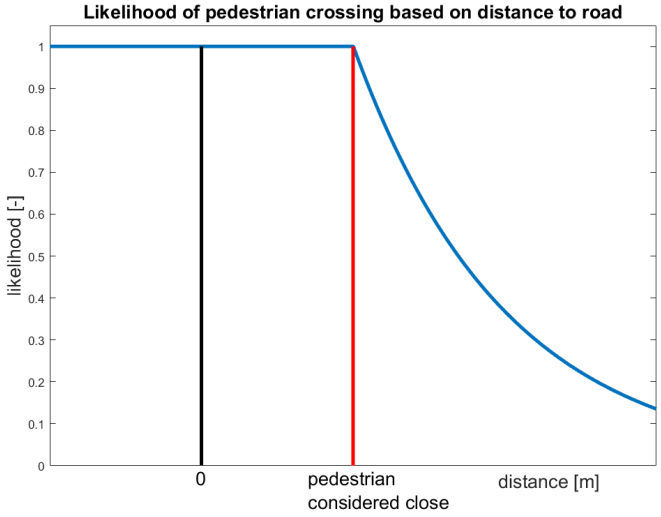



**Figure 5.5:** Distribution function for distance to the road.

### Complete closeness of the pedestrian to the zebra crossing

**Inputs:**

- Distribution for the total distance of the pedestrian (*pedCloseTotal*),
- Distribution for the current and previous pedestrian closeness to the road (*pedCloseRoad*),
- Distribution for the pedestrian closeness to the zebra crossing in a parallel direction to the road (*pedClose*).

This distribution expresses the fact whether, the pedestrian is close to the zebra, or not (later denoted as *pedCloseZebra*). It combines distributions *pedCloseTotal*, *pedCloseRoad* and *pedClose* in conditional probability distribution shown in Table 5.2. As we can see, it is not possible to reach some states in the table. Pedestrian can not be close in total while being far from road and from zebra and it is not even possible for the pedestrian to be far from the crossing in total while being close both to road and zebra.

| pedCloseTotal | pedCloseRoad | pedClose | P(pedCloseZebra) |
|---|---|---|---|
| true | true | true | 1.0 |
| true | true | false | 0.8 |
| true | false | true | 0.7 |
| true | false | false | not possible |
| false | true | true | not possible |
| false | true | false | 0.3 |
| false | false | true | 0.3 |
| false | false | false | 0.0 |

**Table 5.2:** Distribution for complete closeness of pedestrian to the zebra crossing.

### Likelihood of the pedestrian crossing the street given the movement angle relative to the zebra crossing

**Inputs:**

- probability that the current angle is suitable for crossing,
- probability that the previous angle is suitable for crossing.

The distributions for probabilities that the current/previous angle is suitable for crossing is based on the difference of its direction from direction needed for cross the street (later denoted as *pedTowardsZebra*). It has a given threshold where the angle is considered to be close enough and therefore the likelihood is 1.0. Afterwards this distribution decreases exponentially. The likelihood is shown in Figure 5.6.

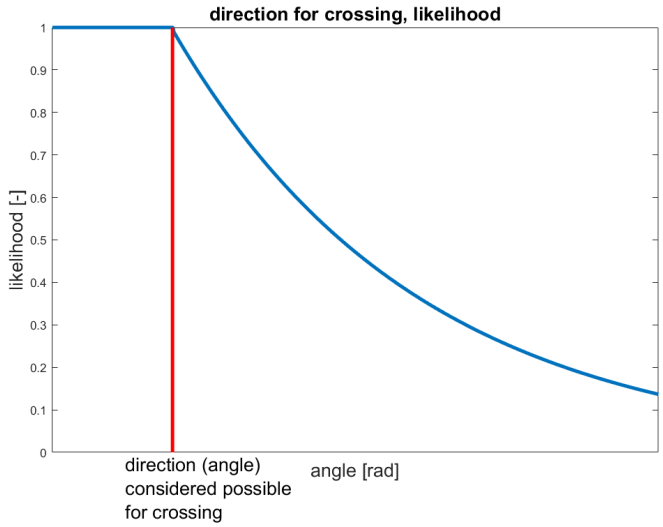



**Figure 5.6:** Distribution function of crossing based on angle (of direction)

The final distribution of likelihood depends on probabilities of crossing (based on angle of pedestrian) in the current and previous time step. This distribution is described in Table 5.3.

| current | previous | P(pedTowardsZebra2) |
|---|---|---|
| true | true | 1.0 |
| true | false | 0.7 |
| false | true | 0.4 |
| false | false | 0.1 |

**Table 5.3:** Distribution for probability that the pedestrian is crossing the street given its angle with respect to the zebra crossing.

### Distribution for pedestrian currently crossing the street

**Input:**

- distance of the pedestrian to the road, measured in metres.

This distribution models the likelihood that the pedestrian is crossing the street at the moment (later denoted as *isCrossing*). It is based on the fact that if pedestrian's distance to the road is less than 0 it means, that pedestrian is already crossing the road. Distribution has likelihood of 1.0 for values less than 0 and than decreases exponentially rapidly (shown in Figure 5.7).

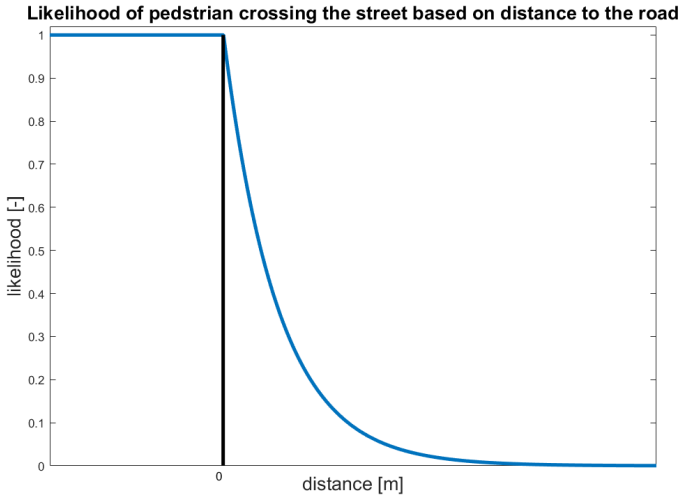



**Figure 5.7:** Distribution function for pedestrian crossing based on the distance to the road.

### Complete distribution for the pedestrian crossing the road

**Inputs:**

- Distribution for pedestrian currently crossing the road (*isCrossing*, 5.1.2),
- Complete closeness of pedestrian to the zebra crossing (*pedCloseZebra*, 5.1.2),

- Likelihood of pedestrian crossing the road given its movement angle relative to the zebra crossing (*pedTowardsZebra*, 5.1.2).

This final distribution expresses whether the pedestrian intends to cross the road, or not (later denoted as *pedCrossing*). It combines three previously described distributions in conditional probability distribution. This distribution is shown in Table 5.4. In the case, in which the pedestrian is already crossing the road, the likelihood of the crossing is always 1.0, no matter what the values of *pedCloseZebra* or *pedTowardsZebra* are.

| isCrossing | pedCloseZebra | pedTowardsZebra | P(pedCrossing) |
|---|---|---|---|
| true | any | any | 1.0 |
| false | true | true | 0.9 |
| false | true | false | 0.7 |
| false | false | true | 0.5 |
| false | false | false | 0.0 |

**Table 5.4:** Table for pedClose3 distribution

## 5.2 Data preprocessing

The input data received from UP-Drive car sensors are already processed (e.g. object detection). However for the purpose of our research, we need to process these data. The measured data contain information about all detected objects such as pedestrians, cars, bicycles, trucks, etc.

The first step of the data preprocessing is the filtering of the input information about EGO car and also about the other detected objects. If we are working only with pedestrians, we can extract easily the information only about them based on the object's classification.

Apart from data received from EGO car, there is a map with roads, junctions and zebras where does the EGO car moves.

### 5.2.1 Observed data about EGO car

The EGO car contains many different sensors (e.g. camera, lidar, radar). However, due to the UP-Drive confidentiality, we will not describe these sensors in detail.
As the most important measured values, we have position of the car in the map based on GPS coordinates, speed of the car, its acceleration and direction. We can see that we do not receive all the information that we use later in the decision system. For example, we do not have an information about EGO car distance to the zebra. Since we have information about position of EGO car in the global coordinates and we know coordinates of the next zebra crossing, we can compute this distance.

More problems with measured data arise with other objects than EGO car. In the next section, we will describe the received data about pedestrians and the way we process

them.

### 5.2.2 Pedestrian data

Each measured value has a flag indicating whether the measured value is correct and therefore can be used. It is a simple 1/0 value.
Each object has its type and an ID. ID number is really useful since we need to track the movement of objects in time.

#### Measured values

As the most important measured values, we have position of the pedestrian measured in meters with respect to the position of EGO car, speed and acceleration of pedestrian and the angle of pedestrian's movement.

We can see a few differences in measuring compared to the measured values about EGO car. For example, the position of pedestrians is measured with respect to the position of EGO car It is not measured in the global coordinates as the position of EGO car is.

**Storing of information**: We need to use information about pedestrian in the previous time step sometimes. For this, we store the information about current and previous time step in HashMap.

The key of this map is an ID of the pedestrian. The type of the stored values is a newly created Java class called `PedestrianData`. It contains two data structures, which store information about current and previous state of the pedestrian. The values stored in these structures are:

- *valid* - boolean value, indicates whether these values are valid, which means that we have information about previous time step,
- time step - time, when the data were measured,
- pedestrian $x$ and $y$ position,
- velocity,
- acceleration,
- angle of direction,
- distance to the nearest zebra in the direction parallel to the road,
- distance to the road,
- distance to the zebra in total.

When we have no information about pedestrian state in the previous time step, *valid* is set to `false`, and we have to estimate the previous state (we set the previous values close to the values in the current time step).

### 5.2.3 Data processing

There is a lot of needed information that is not measured. We need to compute pedestrian distance to the zebra crossing in the direction parallel to the road, distance to the road, total distance to the edge of zebra crossing and a few more (these distances are described in Section 5.1.1). We will show you how we compute these needed values.

**Pedestrian distance to the zebra crossing**: As we have already mentioned before (Section 5.1.1), it is a distance of the pedestrian to the zebra crossing in the direction parallel to the road. It is denoted as $D_z$ in Figure 5.2. Since values about pedestrian's position are measured with respect to EGO car, we need to compute with the distance of the EGO car.

**Pedestrian distance to the road**: The situation with pedestrian distance to the road is quite similar to the previous point. We use $(x, y)$ position of the pedestrian and the direction of car specified by its yaw angle. Since we know this angle, we can easily construct a vector as

$$\vec{v} = (v_x, v_y) = (\cos(yaw), \sin(yaw)), \tag{5.1}$$

where $yaw$ is the yaw angle.
From this angle and the information that the line specified by vector $\vec{v}$ goes through the point $[0, 0]$ (centered at EGO car position), we can create general form of equation for the line $l$

$$l : a \cdot x + b \cdot y + c = v_y \cdot x - v_x \cdot y + 0 = 0. \tag{5.2}$$

If we consider the pedestrian to be at position $P = (P_x, P_y)$, the distance to the line given by the vector $\vec{v}$ (Equation 5.1) with general form of equation of a line $l$ (Equation 5.2) is

$$d_{orthogonal}(P, l) = \frac{|a \cdot P_x + b \cdot P_y + c|}{\sqrt{a^2 + b^2}}. \tag{5.3}$$

The last step is to determine pedestrian distance to the road. It is denoted as $D_r$ in Figure 5.2. Since the computed distance is the 'distance to the car' in orthogonal direction (Equation 5.3). Therefore, we get pedestrian distance $d_{road}$ to the road as

$$d_{road} = d_{orthogonal}(P, l) - d_{car2edge}, \tag{5.4}$$

where $d_{car2edge}$ is distance of the car to the edge of the road.

**Total distance of the pedestrian to the zebra crossing**: The total distance of the pedestrian to the zebra crossing is computed easily from its distance to the zebra in the direction parallel to the road ($d_{parallel}$) and from pedestrian distance to the road ($d_{road}$). It is denoted as $D_t$ in Figure 5.2. We compute the total distance as:

$$d_{total} = d_{road} - d_{parallel}. \tag{5.5}$$

**Angle of pedestrian movement**: We have decided not to use the values of measured yaw angle of pedestrian movement because there was a lot of values that did not reflected the real state well.
We decided to compute this angle from the position of pedestrian in the previous time

step. We started with computing $\Delta x$ and $\Delta y$. Once we had these values, we could compute the angle (called *pedAngle*) as

$$pedAngle = \text{atan2}\frac{\Delta y}{\Delta x}, \tag{5.6}$$

where atan2 is a function defined as in Equation 5.7.

$$\text{atan2}(y,x) = \begin{cases} \arctan\frac{y}{x} & \text{if } x > 0 \\ \arctan\frac{y}{x} + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan\frac{y}{x} - \pi & \text{if } x < 0 \text{ and } y < 0 \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0 \end{cases} \tag{5.7}$$

When the pedestrian is standing on the spot (thus $\Delta y = 0$ and $\Delta x = 0$), or we do not know its previous position, we use the measured angle.

When we do have the information about the pedestrian movement direction we can compute its difference to the direction of the zebra crossing. We already described it in Sections 5.1.1 and 5.2.3. The value of this angle is the difference between the current direction (vector) of the pedestrian movement and the direction of zebra. It is illustrated in Figure 5.3.

## 5.3 Prediction of the future intentions

We studied and proposed the method estimating the pedestrian intention and predicting the pedestrian motion in a one, two, and three seconds time horizon.

Understanding the current traffic situation is a crucial intermediate step on the way towards a self-driving car. The dynamic objects motion prediction based solely on physical laws (inertia) does not suffice. The intentions of other dynamic objects, traffic participants, has to be estimated and taken into account in the reasoning module.

We need to predict the pedestrian motion to be able to predict their future intentions. Having it, we can compute estimate the probability that the pedestrian wants to cross the road.

### 5.3.1 Motion prediction

We used *physics-based* models for our motion prediction. *Physics-based* motion models are the simplest ones, which consider that the motion of vehicles depends on the laws of physics only. They can predict reliably the motion of other traffic participants for up to 1 second time horizon. Physics-based models have been the most widely used.

Correct short-term prediction of pedestrian motion is an important issue when dealing with pedestrian safety in traffic situations. We used a simple physics-based prediction, which explores the previous position of the pedestrians. The presented motion prediction consists of two steps:

- filtration of the input data;
- linear regression of the motion.

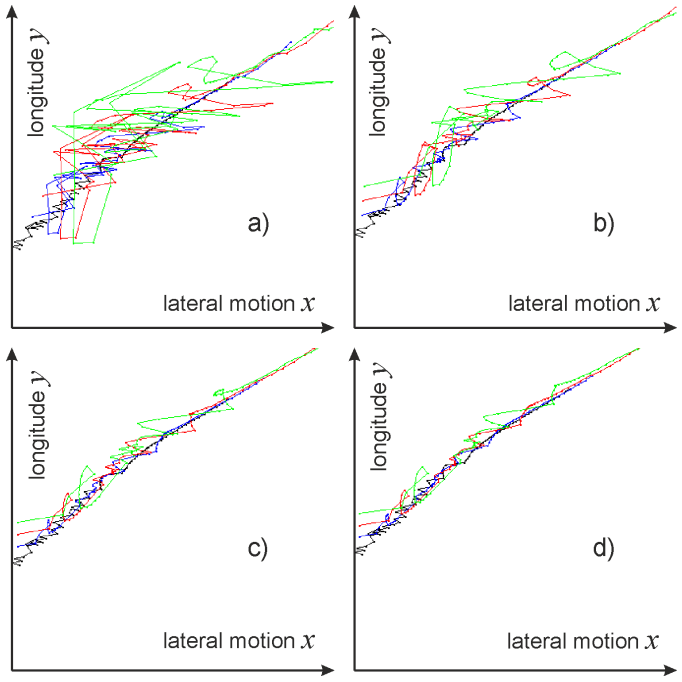



**Figure 5.8:** Motion of a pedestrian: a) Without filtering; Cubic filtering with previous positions gathered from: b) 1 second window; c) 2 seconds window, d) 3 seconds window into the past.

The filtering step was used for smoothing out a noisy input data. Several different filtering methods were applied: average, linear, exponential, quadratic, and cubic. The input data from several previous measured steps were filtered and subsequently the linear regression was used for estimating the future position of the pedestrian. The filtering was performed with different time window size (one, two and three seconds from the past) on previous measured data. The visualization of the results of cubic filtering and predictions for different window sizes is shown in Fig. 5.8, where the $y$-axis represents the longitudinal and $x$-axis represent the lateral position of the pedestrian. The curve itself is parametrized by time. The whole curve corresponds to about 8 seconds duration. The sampling is every 0.1 seconds. The black curve represents the actual motion of the pedestrian, the blue curve represents the motion prediction in time $t + 1$ second, the red curve represents the prediction in time $t + 2$ seconds, and the green curve represents the prediction in time $t + 3$ seconds.

The motion prediction was developed as a part of UP-Drive project and was made by Ing. Miroslav Uller.

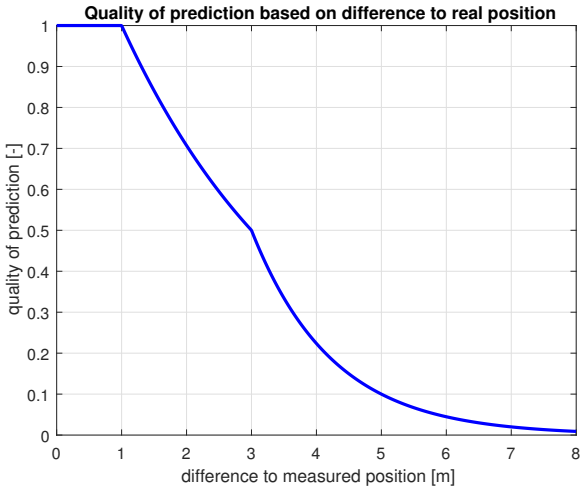



**Figure 5.9:** Curve for estimating the prediction quality.

The quality of the predicted pedestrian position was determined based on the difference between the predicted position and the real measured position in time $t_i$ given in meters. The curve used to evaluate the prediction quality is shown in Fig. 5.9. Two requirements on the quality of the prediction that were used in this application are:

1. If the difference of the predicted pedestrian's position and the real measured position in time $t_i$ is lower than one meter, this prediction is considered as successful.

2. The distance between one and three meters is decreasing exponentially and slowly. The three-meter break point is based on the width of the road line.

| Method | Time prediction | | |
|---|---|---|---|
| | 1 second | 2 seconds | 3 seconds |
| | Quality [%] | Quality [%] | Quality [%] |
| Average | 62.9 | 53.0 | 42.8 |
| Linear | 80.1 | 70.1 | 61.1 |
| Exponential | 83.5 | 73.5 | 64.6 |
| Without smoothing | 91.6 | 80.5 | 68.8 |
| Quadratic | 88.5 | 79.1 | 70.0 |
| **Cubic** | **91.8** | **83.6** | **74.3** |

**Table 5.5:** The success rate of the motion prediction for various data filtering methods.

The prediction of the motion was set for three different time intervals from the current time $t$, i.e.: $t+1$, $t+2$ and $t+3$ seconds. The obtained results are presented in Table 5.5, which shows individual filtering methods and the corresponding prediction results of the future position. The results are listed from the lowest achieved score for time $t+3s$. The absolute prediction deviations of used filtering methods in considered time intervals are shown in Fig 5.10.

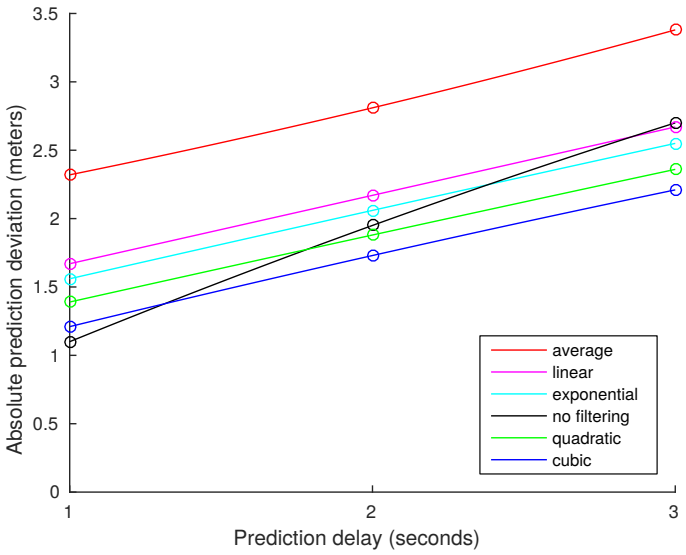



**Figure 5.10:** Absolute motion prediction deviation.

As it can be seen, no filtering has the smallest deviation for the one second prediction. However, the prediction worsens in the next two time instances drastically. The cubic filtering has only slightly worse prediction error in the first second, but it showed the best absolute prediction for the remaining time instances as well as the best overall prediction quality. Based on these results, we have decided to use the cubic filtering.

### 5.3.2 Future intensions estimation

The individual probabilities of pedestrian intention to cross the road were computed for each motion prediction estimated 1, 2, and 3 seconds into the future. Consequently, the overall probability of pedestrian wanting to cross the road was computed as a weighted sum of the probabilities for each time instance.

The weights of overall probability are qualities of motion prediction of chosen method. These values are shown in table 5.5. We denote them as $w_{t+1}$, $w_{t+2}$, $w_{t+3}$. The overall probability is computed from computed probabilities in times $t$, $t+1$, $t+2$, $t+3$ (corresponding probabilities are called $P_t$, $P_{t+1}$, $P_{t+2}$, $P_{t+3}$) and weights $w_{t+1}$, $w_{t+2}$, $w_{t+3}$ can be described by algorithm:

1: initialization $P_{overall} \leftarrow P_t$
2: **if** $P_{overall} < P_{t+1}$ **then**
3: $\quad P_{overall} \leftarrow P_{overall} + (P_{t+1} - P_{overall}) \cdot w_{t+1}$
4: **end if**
5: **if** $P_{overall} < P_{t+2}$ **then**
6: $\quad P_{overall} \leftarrow P_{overall} + (P_{t+2} - P_{overall}) \cdot w_{t+2}$
7: **end if**
8: **if** $P_{overall} < P_{t+3}$ **then**

9: $P_{overall} \leftarrow P_{overall} + (P_{t+3} - P_{overall}) \cdot w_{t+3}$
10: **end if**
11: **return** $P_{overall}$

We modify the value of $P_{overall}$ only if the value of $P_{t+n}$, where $n = \{1, 2, 3\}$, is larger than $P_{overall}$. In case that $P_{t+n}$ is larger than $P_{overall}$, we add to $P_{overall}$ the difference between $P_{t+n}$ and $P_{overall}$ weighted by $w_{t+n}$. We can write this update as follows:

$$P_{overall} = P_{overall} + (P_{t+n} - P_{overall}) \cdot w_{t+n}. \tag{5.8}$$

With this approach, the predicted probability $P_{t+n}$ cannot decrease the current overall probability. Otherwise we could have the situation when the pedestrian is at the current time $t$ certainly crossing the road (probability 1.0) but we predict that in time $t + 3$ the pedestrian will already be far from zebra crossing and therefore much unlikely crossing the road. In this situation, the probability of crossing in $t + 3$ would greatly affect the current overall probability and would not reflect pedestrian's intentions correctly.

The proposed approach of computing overall probability predicts the probability that the pedestrian wants to cross the road in the next three seconds time, i.e. in time interval $\langle t, t + 3\rangle$.

### 5.3.3 Evaluation of results

We divided evaluation of results into two main parts:

1. Influence of motion prediction accuracy on the pedestrian intention estimation.
2. Accuracy of computed overall probability of pedestrian intention.

#### Influence of motion prediction accuracy on the pedestrian intention estimation

We evaluated the motion prediction capabilities of methods described in this contribution. The motion predictions were computed for three instants ahead of the current time: one, two and three seconds. The results for the average success rate of the predictions are presented in Table 5.5. It can be seen that the one second ahead prediction using the cubic filtering achieved 91.8% success rate. The success rate for two and three seconds predictions decreased. This is an expected result because of physics-based motion estimation properties.

The median deviation of predicted position and measured prediction for the cubic filtering case was 1.21 meter. This was the best value obtained when compared to the other tested filtering methods. This simple prediction can be improved by using more complex systems such as maneuver-based or interaction-aware motion models. However, these models are more time consuming, which is their disadvantage in comparison to the physics-based models.

| Time scope (seconds) | Motion prediction error (%) | Intention prediction error (%) |
|---|---|---|
| $t+1$ | 8.2 | 15.4 |
| $t+2$ | 16.5 | 17.4 |
| $t+3$ | 25.7 | 17.7 |

**Table 5.6:** Consistency of predictions.

The correct intention estimation of other traffic participants is closely related to their motion prediction. Thus, we focused on the influence of decreasing success of motion prediction on the pedestrian intention estimation. The average error rates for motion prediction and intention estimation for different time intervals are shown in Table 5.6. Values in this table are average differences between predictions of probability made in time $t$ (this means $1, 2$ and $3$ seconds into the future) and the real values in probability in these times. The relationship between motion prediction and intention estimation error rate is displayed in Figure 5.11. The dotted line represents a trend line that should be followed by linearly dependent phenomena. As it can be seen, the relationship between motion

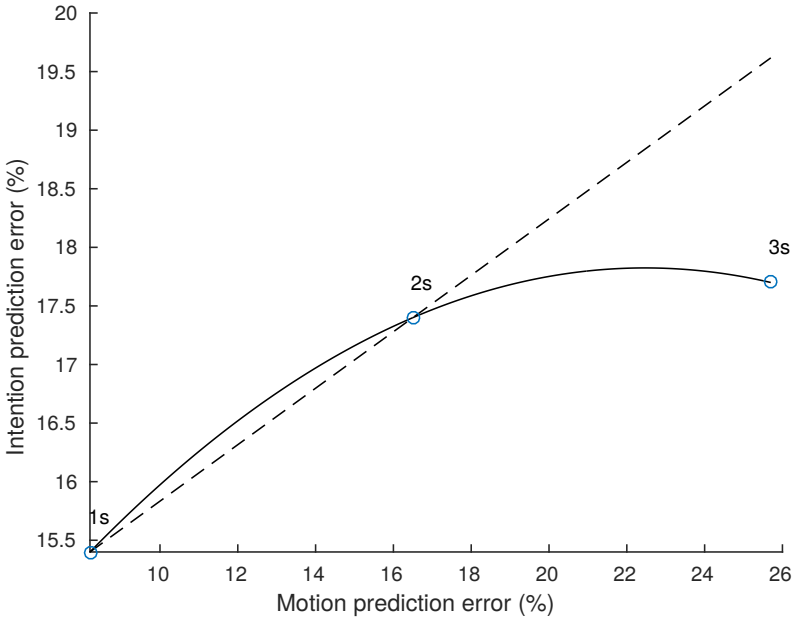



**Figure 5.11:** The relationship between motion prediction error and the intention estimation error.

prediction and intention estimation error is not linear. It is in fact sub-linear, i.e. a lower accuracy of motion prediction does not affect the accuracy of intention estimation to a large extent. Since the relationship between motion prediction and intention estimation is not linear, we can focus largely on improvements in intention estimation independently of the motion prediction.

### Accuracy of computed overall probability

In the previous section, we have observed that the lower accuracy of motion prediction does not affect the accuracy of intention estimation to a large extent. In this section, we study the overall success of intention estimation prediction.

In this evaluation, we compared the overall probability $P_{overall}(t)$ in time step $t$ to the highest probability in the interval $\langle t, t+\tau \rangle$, where $\tau \in \langle 1s, 3s \rangle$ with $0.1s$ step. The interval for $\tau$ is derived from the facts that we predict pedestrian intention 1, 2, and 3 seconds ahead and we have data measured every 0.1s, therefore we cannot choose smaller step. The computation of prediction accuracy is shown in algorithm below, where $P_{real}$ is a real computed probability:

1: $sum \leftarrow 0.0$
2: **for** all *pedestrians* **do**
3: **for** every time step $t$ **do**
4: $P_{max} \leftarrow$ max_probability($P_{real}$ in $\langle t, t+\tau \rangle$)
5: $diff \leftarrow |P_{max} - P_{overall}(t)|$
6: $sum \leftarrow sum + diff$
7: **end for**
8: **end for**
9: $diff_{average} = \frac{sum}{\text{number of measurements}}$
10: **return** $diff_{average}$

For every pedestrian and every time step $t$ we first find the maximal probability $P_{max}$ in interval $\langle t, t+\tau \rangle$, where $\tau \in \langle 1s, 3s \rangle$ with $0.1s$ step. Value of $\tau$ stands for window size is changed after the whole evaluation. Second, we compute the absolute difference between predicted overall probability $P_{overall}$ and real maximal probability $P_{max}$. Than we do add this difference to the *sum* of all differences. In the end we divide this *sum* by number of measurements and we obtain an average prediction error.

Obtained results for different window size $\tau$ are shown in Figure 5.12 with prediction errors in times $t+1, t+2$ and $t+3$s shown in table 5.7. These errors are chosen because we do predict intention estimation $1, 2$ and 3 seconds forward.

| **Time scope (seconds)** | **Intention prediction error (%)** |
|---|---|
| $t+1$ | 5.53 |
| $t+2$ | 6.65 |
| $t+3$ | 7.16 |

**Table 5.7:** Prediction error

As it can be seen, the error for overall probability that is computed as a linear combination of probabilities $P_t, P_{t+1}, P_{t+2}, P_{t+3}$ with coeficients $(1, w_{t+1}, w_{t+2}, w_{t+3})$ is much lower than the error of $P_t$ alone (as it can be seen in Table 5.6). We can see that the relationship between the window size and the intention estimation error is, once

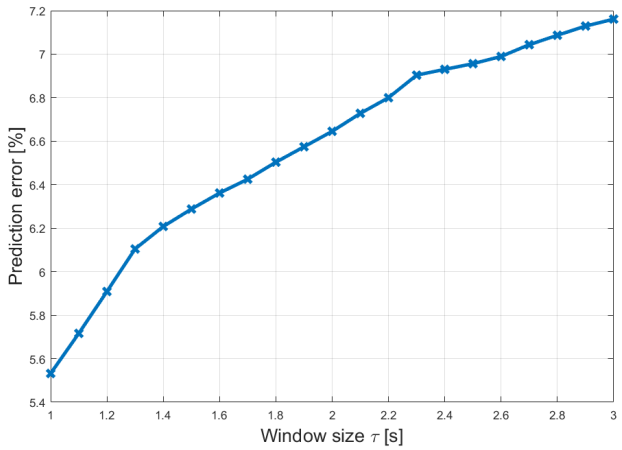



**Figure 5.12:** The relationship between window size and the intention estimation error.

again, not linear. We can see that the intention estimation error increases with the size of window but the speed of error growth decreases with the window size. It is, of course, result of the relationship between motion prediction error and the intention estimation error not being linear as well (as shown in Figure 5.11) and it corresponds with results obtained in Section 5.3.3.

## 5.4 Visualization

We needed to visualize the results from our Bayesian network. For this, we use a very simple and low-level visualizer developed for visualizing simulation data. It is written in Java and uses JOGL (Java OpenGL Library). It was developed for the use in the UP-Drive project by Miroslav Uller.
The visualizer renders a collection of renderable objects. There are several renderable objects implemented:

- point - used for pedestrians,
- rect - rectangle, used for EGO car,
- line - used for visualizing pedestrians' direction of movement,
- TextLabel - a text label, used for displaying current probabilities of pedestrian crossing the road.

EGO car is at position $[0, 0]$ and is represented by a picture of a car. Bellow the car, we can see the number of current time step and information about its current state - speed and acceleration.

Pedestrians are represented by colored squares and the direction of their movement is represented by a blue line. The color of pedestrian shows how probably is this pedestrian

to cross the road. Colors are from blue (not likely to cross) to red (very likely to cross the road). Next to each pedestrian, we can see overall probability that pedestrian wants to cross the road, in parentheses is shown probability which takes into account only observations in current time step (previously denoted as $P_t$).

Zebra crossing is in the visualization represented by white stripes. We can see a situation shown in our visualizer in Figure 5.13.

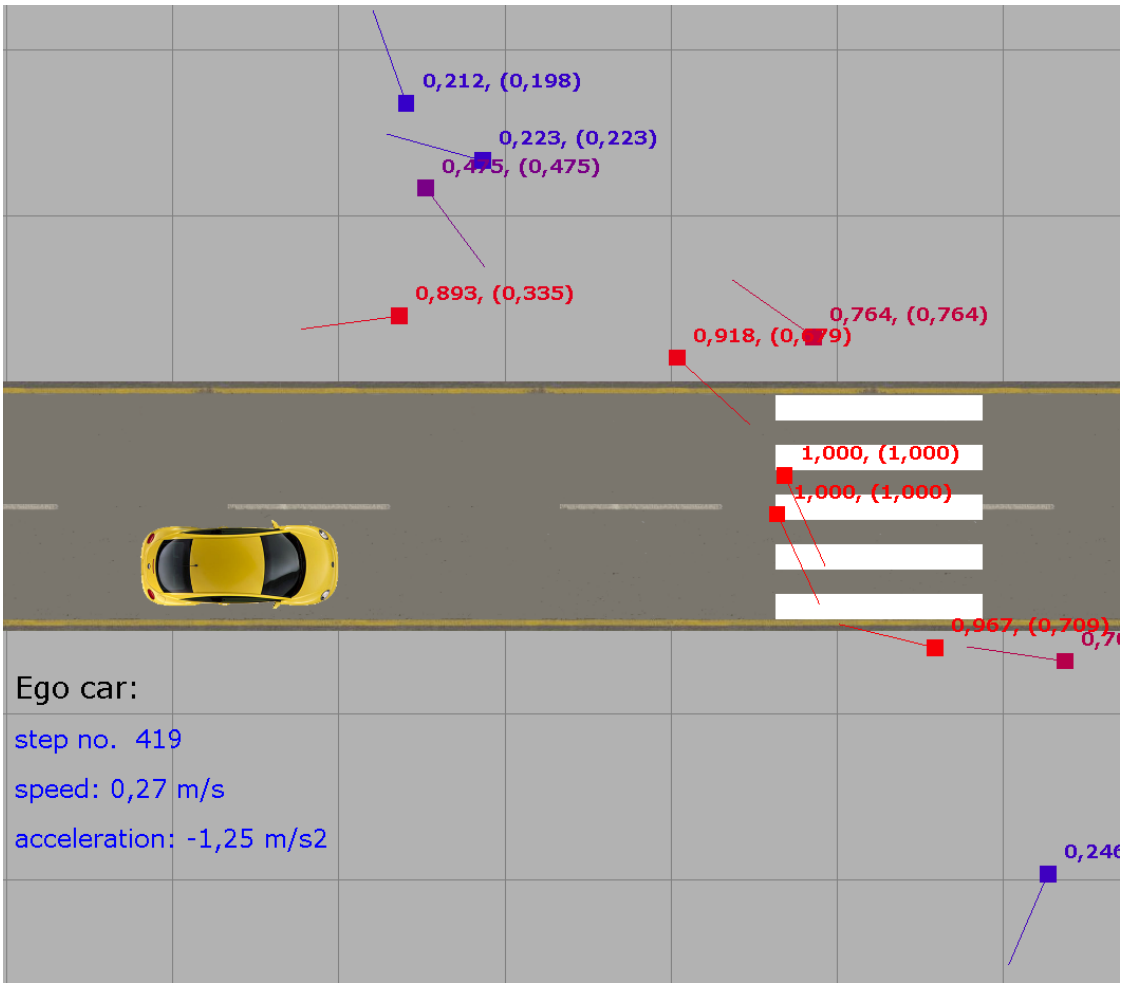



**Figure 5.13:** Example of a visualization.

# Conclusions

This thesis dealt with the intention estimation of traffic participants. For this intention estimation we used Bayesian networks.

At the beginning of this thesis we have sketched the necessary knowledge necessary to understand Bayesian networks. We followed by describing probabilistic programming in general and programming language called Figaro in particular.

Afterwards, we explained the needed parts and implementation of the simulator that provides similar data as a self-driving car. We implemented this simulator in Java and used it for testing Bayesian network for general traffic situations.

The main goal of this thesis was to design and implement Bayesian network capable of predicting the intentions of the traffic participants. We designed two Bayesian networks. The first network was designed and tested with the data provided by created simulator. This network was a more general one.

The second network was designed for intention estimation of pedestrians near the zebra crossing. This Bayesian network used real measured data. We provided prediction of future estimation of pedestrians along with the motion prediction which is vital for an accurate intention prediction. Since we had real measured data for use with this Bayesian network we studied influence of motion prediction accuracy and accuracy of predicted pedestrian intentions.

The motion prediction error of our used approach was 8.82% for predicting one second ahead, 16.5% for predicting two seconds ahead and 25.7% for predicting three seconds ahead. We found out that the relationship between motion prediction and intention estimation is not linear, so we can focus largely on improvements in intention estimation independently of the motion prediction in the future work.

We evaluated accuracy of computed predicted overall probability of pedestrian intention. The proposed approach computed the probability that the pedestrian wants to cross the street in the next three seconds time, i.e. in time interval $\langle t, t+3 \rangle$. By comparing predicted overall probability with real values of probability in the future, the error for prediction 1 second ahead was 5.53%, 2 seconds ahead was 6.65% and for 3 seconds ahead the error of prediction was 7.16%. We found that the intention estimation error increases

with the size of the prediction window but the speed of error's growth decreases with the window size.

We can state that all the goals of the bachelor thesis have been fulfilled and even extended by design of the Bayesian network for intention estimation of pedestrians near the zebra crossing and by using real measured data.

One of the possible future improvements of the topics in this thesis is an improvement of motion prediction of pedestrians, we can use for example Kalman filtering [13], [1]. In this thesis, we suppose the movement of each pedestrian to be independent of to the movement of other pedestrians. However, in real situation this independence does not always hold true and therefore taking the movement of other pedestrians into account might lead to better estimation of pedestrian intentions and can be one of the following areas of research.

# Bibliography


[1] D. Ellis, E. Sommerlade, and I. Reid. Modelling pedestrian trajectory patterns with gaussian processes. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1229–1234, Sept 2009.

[2] J. Ziegler et al. Making bertha drive an autonomous journey on a historic route. *IEEE Intelligent Transportation Systems Magazine*, 6(2):8–20, Summer 2014.

[3] Z. Ghahramani. Introduction to Graphical Models, 2007.

[4] Ch. Katrakazas, M. Quddus, W. Chen, and L. Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60:416 – 442, 2015.

[5] S. Kokoska and D. Zwillinger. Probability. In *CRC Standard Probability and Statistics Tables and Formulae*, pages 19–38. CRC Press, 2000-03.

[6] V. Mihajlovic and M. Petkovic. Dynamic Bayesian Networks: A State of the Art, 2001. DMW-project.

[7] R. E. Neapolitan. Learning bayesian networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, New York, NY, USA, 2007. ACM.

[8] M. Paskin. A Short Course on Graphical Models, 2. Structured Representations. `http://ai.stanford.edu/~paskin/gm-short-course/`, 2003. [ONLINE], Accessed: November, 2016.

[9] A. Pfeffer. *Practical Probabilistic Programming*. Manning Publications Co., 1. edition, 2016.

[10] D. Pomerleau and T. Jochem. Rapidly adapting machine vision for automated vehicle steering. *IEEE Expert*, 11(2):19–27, Apr 1996.

[11] S. Rao and D. Tse. Discrete Mathematics and Probability Theory, note 11, 2009.

[12] T. A. Stephenson. An introduction to Bayesian network theory and usage. Technical report, IDIAP, 2000.

[13] G. Welch and G. Bishop. An Introduction to the Kalman Filter: SIGGRAPH 2001 Course 8. In *Computer Graphics, Annual Conference on Computer Graphics & Interactive Techniques*, pages 12–17, 2001.

[14] WHO. Global status report on road safety: supporting a decade of action, 2013. World Health Organization report.