# Probabilistic Motion and Intention Prediction for Autonomous Vehicles

**Probabilistische Bewegungs- und Intentionsvorausssge für autonome Fahrzeuge**

Master-Thesis von Lina Jukonyte aus Utena, Litauen

Juli 2019

TECHNISCHE UNIVERSITÄT DARMSTADT

Probabilistic Motion and Intention Prediction for Autonomous Vehicles
Probabilistische Bewegungs- und Intentionsvorausssge für autonome Fahrzeuge

Vorgelegte Master-Thesis von Lina Jukonyte aus Utena, Litauen

1. Gutachten: Prof. Jan Peters, Ph.D
2. Gutachten: Dorothea Koert, M.Sc., Joni Pajarinen, D.Sc. (Tech.)
3. Gutachten: Prof. Matthias Hollick, Ph.D

Tag der Einreichung:

# Erklärung zur Master-Thesis

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Lina Jukonyte, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

In the submitted thesis the written copies and the electronic version are identical in content.

Datum / Date: Unterschrift / Signature:

_____ _____

# Abstract

In autonomous driving observing and evaluating the intentions of other traffic participants is one of the most crucial tasks to ensure safe trajectory planning and appropriate reactions. To be able to make proper judgments of future movement for other traffic participants, various variables must be considered, one of them is current and past positions of the objects of interest. Without proper prediction model, this is a very hard task to do. Usually, intentions are one variable which is not known, but it can be evaluated through partial observations and prior information about the behaviour of the object of interest. Probabilistic models can be very beneficial in this scenario due to their capability to capture variance in motions. Therefore, this thesis presents a probabilistic approach for intention prediction based on trajectory data. To be able to predict motion intentions and direction future steps in trajectory, we propose a probabilistic intention prediction algorithm. During the algorithm training process, we modelled the initial probabilistic model from predefined movement demonstrations. The proposed algorithm was evaluated in a simulated driving environment and on data of a real car. In the simulated environment, two maps environments were used and prior knowledge, such us probabilities to go to one or another direction, which we have before starting to move. Prior information was incorporated into the algorithm regarding the particular map setting and prior demonstrations. As proof of concept, various experiments were done. They demonstrated that the proposed algorithm can correctly predict future movement intention ahead of the time: algorithm correctly predicts movements and further steps within $30-40\%$ of testing trajectory, while the easiest recognizing movement to left direction. While testing an experimental setup with incorporated prior information, the algorithm predicts intentions much faster.

# Zusammenfassung

TODO

# Acknowledgments

TODO

# Contents

# Figures and Tables

## List of Figures

## List of Tables

# Abbreviations, Symbols and Operators

# 1 Introduction

## 1.1 Motivation

Recent years were full of massive developments towards autonomous driving in automotive industry. Achievements in one area can be helpful in developing other areas, i.e. great success in image recognition and perception can allow computers to achieve super-human performance [6]. Unfortunately, image recognition and environmental perception alone are not enough to solve all the problems which autonomous cars are facing. In ideal circumstances achieving full autonomy in the cars would help not only to save the environment, but it would be beneficial for other participants as well, since autonomous cars would bring more smoothness and safety to the roads. Industrial innovation experts from ARK Invest strongly believe that with fully autonomous cars, accidents on the road would drop to 80% [7].
Although autonomous cars have engaged a wide range of engineering disciplines for some time already, this area not fully developed yet and will continue to engage engineers even more in the future. At the moment one of the biggest achievements, which is equipped into the majority of new cars is Automotive Driver Assistance Systems (ADAS), which does not enable yet full autonomy of the cars, but it successfully assists driver while driving a car.
The critical part of driving, whereas it is driver included or driverless car, is interaction with other traffic participants. Being not alone on the road requires the ability to evaluate situations and predict the intentions of other drivers on the cars. For human, this task does not seem so complicated, since they can see the environment, traffic signs, lights, turning signals, the situation on the road, etc. and act on these intuitively. For a driverless car, this task is a bit more complicated since there are a lot of information vectors which need to be considered. But to be able to function properly algorithms of the car should be able to predict the future action and some trajectory in advance. To make these kinds of predictions, the algorithms should get and use all information which is available from sensors and which was stored while driving (past observations of other cars). Additionally the algorithm should use information which was used to teach the algorithm how to separate one action from another or/and additional collaborations with infrastructures. Prediction making is a difficult task, which is easier to solve having and understanding information about the possible behavior of other traffic users [8].
For motion and intention prediction, an algorithm must predict the intention or the goal of another traffic user and to predict motion which possibly will be made in order to achieve a predicted goal [9, 10]. To make this type of prediction, a general approach is to consider each movement class separately and to include prior information about each movement class into prediction making process. Lately, there were several methods proposed for motion and intention prediction: Gaussian Mixture Model (GMM) [11], Dynamic Bayesian Networks (DBN) and Bayesian Networks (BN) in respectively in [12] and [13]. Hidden Markov Model (HMM) [14, 15], Probabilistic Dynamic Movement Primitive (PDMP) [16], ProMPs [17] and others, more information in the next Chapter. While designing implementation model for our probabilistic intention prediction algorithm, we chose it to be based on Gaussian. Differently from the most other methods, using Gaussian distribution is useful when distribution of values is now known and it does not require information about the final trajectory goal, which is hard to predict while driving. As well, Gaussian distribution are able to give good results even with noisy measurements.

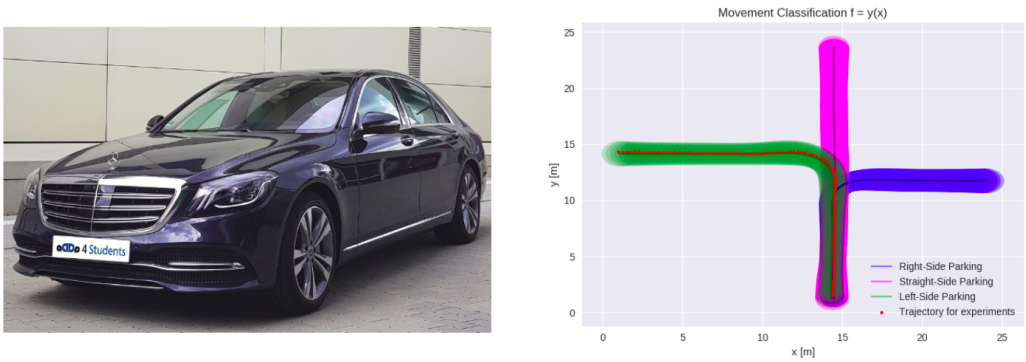


**Figure 1.1:** The left side shows the aDDa car, the right side illustrates the results of one of the experiments cases when the prediction, where the car is going, was made. Hereby the probabilistic model considers also the variance in the demonstrated trajectories, depicted in green, blue and pink

Even though to understand human behavior is a crucial task for further development of autonomous car, humans are very irrational and unpredictable and due to that, it is very hard to model them. However, in this document, we present a probabilistic intention prediction algorithm to predict motion and intentions from early observation for autonomous cars. To achieve this goal, the algorithm first learns the examples for all possible movement classes. When a car starts to move, it collects observations of the current position and predicts what intention for future movement is. Considering that the most important thing while driving is safety. We did a literature review which are the most common attacks on sensors (which information is using for prediction making) of autonomous car and made suggestions on how to protect the system from being attacked, as well as described the main privacy issues.
Probabilistic intention prediction algorithm was developed using two programming languages: Python and C++. The experiments were modeled and ran in ROS environment. ROS - an easy to use framework for developing robot software. In order to simplify the task for creating a complex robot model, it contains a lot of libraries and tools from a lot of robotic platforms. Additionally it has a wide community of users who provide big help while solving problems [18]. RViz was used as ROS simulation visualization tool.
Beyond big giants like Tesla, Google, Aptiv, etc., chasing the dream of driverless cars, big number of academic institutions are also carrying out research in this field. University of Darmstadt (TuD) has its own aDDa working group [19]. aDDa initiative started on February of 2017. 8 different departments at TuD are working for one purpose to develop fully autonomous car by themselves, here, at University. All participants of the working group closely cooperate bringing together interdisciplinary know-how experience to jointly set up and operate an autonomous vehicle. One special feature of aDDa is that the main work is done in the context of student projects (final thesis, semester work, permanent work at the team, etc.). By working together on the complex tasks of autonomous driving, participants are solving problems for tomorrow.
This thesis aims to investigate a probabilistic approach for intention prediction based in trajectory data. In particular, the approach is also tested with trajectory data, recorded with the real aDDa car.

## 1.2 Thesis Outline

This thesis focuses on developing and evaluating probabilistic based movement and intention prediction algorithm. The thesis is organized as follows:

- Chapter 2. **Literature Review** focuses on the literature review on the intention prediction.
- Chapter 3. **Probabilistic Trajectory Prediction** describes theoretical tools which help to formalize the problem for intention prediction.
- Chapter 4. **Setup and Implementation** defines how and why simulation was set the way in was. Defines inputs for the system and experiments.
- Chapter 5. **Experiments and Results** describes experiments which were done during the thesis writing period and evaluate results which were received by performing various experiments.
- Chapter 6. **Security Aspects** is based on the fact that "there is no safety without security" and tries to explain the main security and privacy issues of autonomous cars related to movement predictions.
- Chapter 7. **Conclusion and Future Works** wind up this thesis with conclusions and future works based on the findings of previous chapters.

# 2 Literature Review

This chapter introduces the state of the art for prediction making. Explains the most common methods for movement prediction in literature and in practice, outlines its' advantages and disadvantages.

## 2.1 State of the Art for Movement Prediction Making

The most studies related to movement prediction on vehicles focus on lane change predictions. And there are various different methods are proposed to solve this task, the most popular are: DBN, BN, Support Vector Machine (SVM), HMM, Mind-tracing and Fuzzy Logic (FL).
BN could be considered as a graphical representation of probability distribution. In [12, 13] DBN and BN are used to recognize actions which driver is intend to perform. Lateral movement of a vehicle is expressed using BN, having several various nodes for probability. The probability distribution for every node is determined by doing an analysis of driver behaviour in the past while driving. And ultimately, the final prediction is obtained by calculating the probability of a certain movement with respect to the possibility for every node.
In [20, 21] process of driving is described as a set of various different states while driving. When some particular actions appear in particular defined sequence, it is possible to calculate the probability that the state will change to a particular state. In these works likelihood of states shifting is designed using HMM. Authors of [14, 15] expanded their past work using even more realistic test case - they equipped a vehicle with sensors and used received graphical data to get more accurate results. Graphical models together with HMMs and its extensions were trained using the data from experimental driving, seven different driver models were created: passing, changing lanes (to the right or to the left), turning right or left, starting and stopping. The result authors received and presented was "on average, the predictive power of our models is of 1 second before the maneuver starts taking place" [15].
Author of [22] proposed new method for prediction making, which was named Mind-tracing. It is a computational framework which is able to predict possible drivers' intentions. This method is different from others because here different cognitive model versions which include a flood of a possible intention and action is used. Each action and possible intention are compared with a driver's behaviour at the same time. And the closet to the human behaviours is used for the further intentions expression.
[23] introduces FL as an alternative method for modelling behaviour of the driver. The research paper is mainly focused on the process of decision making on lanes of a highway. For getting results a triangular membership function was used, fuzzy rules were defined by observing training procedure and learning from obtained results. The model was developed using actual traffic data. The used model combines the speed and speed difference of the vehicle, the lead and lag gap distances and the remaining distance to the end of the merge lane as input variables. The precision of prediction using the model was higher than using the binary Logit model. The high prediction accuracy received using this model results in prediction accuracy made using this model overall.
[24] tested the validity and accuracy of SVM in movement prediction. After choosing proper hints for movement changing, data recorded by doing test were divided into different groups which were used for training classifier. Predictions were made using current information of the vehicle and classification hints at the current time.
Even though all methods have the same purpose, it is very hard to compare them directly. Results received having measurements in different situations and in different time steps, e.g. one research paper gives prediction two seconds in advance before a lateral position of vehicle's overlaps with the lane border, while other paper gives prediction only one second before crossing the lane. Furthermore, there is no exact definition of *lane crossing* moment, usually, it is the moment when vehicle cross edge of a lane, but in some paper, it is not clear enough.
As it is possible from the results shown in the table the best methods for predicting movement changes is received by using BN and SVM. These two methods are able to predict quite accurate and with a relatively short period of time, what will lead in having more time in advance to decide which action to make.
For further work, any Bayesian filter/classifier could be an acceptable method for examining behaviour of the driver for various reasons:

- Bayesian-based methods can perform well while working with a very big amount of data;
- It gives results with high accuracy from a problem, containing many features. It is needful to include different physical data while modeling and examining drivers' behaviour. Traditional statistical classifiers most likely to be insufficient while processing high dimensional data.

- Bayesian filter/classifier are robust to over-fitting problem and rely on margin maximization instead of finding an edge for prediction directly from the training data.

## 2.2 Movement Prediction

Foresee future moments and trajectories for dynamical objects in traffic scenarios is vital in order to obviate risks which occur on the roads. Prediction despite of short or long term they are, must have sufficient time in advance to avoid traffic situations we, as traffic participants, don't want. In this section, relevant researches for trajectory and movement predictions are introduced.
There is numerous research made on a trajectory and movement predictions with a vehicle as interest on traffic scenarios. [1] suggesting a one way of classifying methods for motion prediction. The main three categories with an increased rate of flexibility were defined: ***physical-based, maneuver-based*** and ***interaction aware***.

- **Physics-based** motion models are the most simple of all categories. It is considered that the movement of vehicles depends only on the laws of physics. A wider description is in subsection 2.2.1.
- **Maneuver-based** motion models are more advanced than physics-based because maneuver-based motion models also consider future movements of a car which also depends on the maneuver which is intended to perform by a driver. A wider description is in subsection 2.2.2.
- **Interaction-aware** motion models take into account consideration connections between maneuvers of the car, as well as rules of the traffic. This method as not so popular as previous ones due its complicity to adapt to the real life scenarios. A wider description is in subsection 2.2.3.

Figure 2.1 summarizes motion models defined in [1].

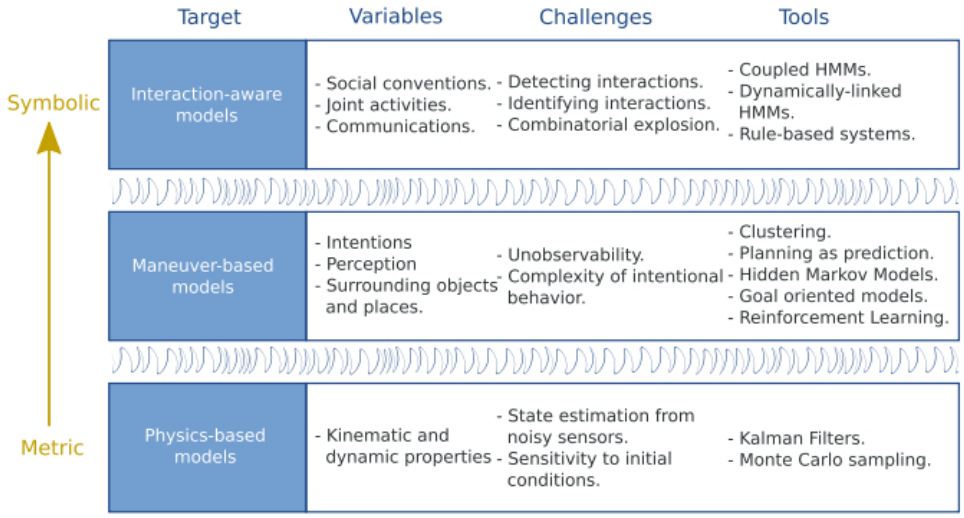


**Figure 2.1:** Motion Modeling Overview [1]

Together with above-mentioned categories, authors of [25] introduced one more category to predict movements - ***data-driven based***.

- **Data-driven** based motion and trajectory prediction can be classified into clustering-based and probabilistic approaches. A wider description is in subsection 2.2.4.

### 2.2.1 Movement Prediction Using Physics-Based Models

Physics-based movement prediction models imply vehicles as a dynamic item, controlled by the physics' laws. Movements are predicted using dynamic and kinematic models with control inputs (e.g. acceleration, deceleration, steering), properties of the car (e.g. length, weight) and some external conditions (e.g. the friction coefficient of the road surface) to the process state of the vehicle (e.g. position, speed, direction). Great work has been done using *physics-based* motion

models and it still remains the most commonly used motion models for motion prediction in the context of road safety. The complexity of the models depends on a representation of the dynamics and kinematics of a vehicle, as well as, how uncertainties are handled, whether or not the road geometry is taken into account, etc.
Dynamic and kinematic models can be used for movement prediction in a lot of different ways, the main difference is how uncertainties are handled. Three main approaches will be described as follow: ***single trajectory simulation, Gaussian noise simulation*** and ***Monte Carlo simulation*** [1].

- **Single Trajectory Simulation.** The simplest method to predict future movements and trajectory of a car is to apply simple dynamic or/and kinematic models for the current state of a car while assuming that the current state of the car is determined with absolutely highest confidence and applied model (dynamic, kinematic or both) is the perfect representation for the movement of the car. This simple approach was used in [26] using dynamic and [27, 28] kinematic models. The main benefit of this straightforward approach is computational efficiency, that allows this method to be used in real time. On the other hand, predictions made by this method do not consider uncertainties of the current state and as a result, predicted movements and trajectories are not trustworthy for use in a long term ( $>$ 1 sec.) predictions.

- **Gaussian Noise Simulation.** The uncertainty of the current state of a vehicle and its evolution during the time is a very important factor in movement or trajectory prediction and it can't be avoided. In [29, 30, 28] this is modelled using a normal distribution. Gaussian Noise function is very popular because of its uncertainty representation in Kalman Filter (KF), which is still a conventional method for vehicle state estimation having noisy sensors measurements in account. There are some cases where dynamic, kinematic and sensor models are linear and uncertainty is modelled using a normal distribution instead of KF Bayesian filter is used. Filtering mainly contains of two steps: prediction and update steps. In the first time step at time step t, a current state of the vehicle is is given to the dynamic or kinematic model, which gives predicted state for the next time step which has a Gaussian distribution shape. In the following step, predicted state of the next time step is combined with sensor measurements of the same time step, which is Gaussian distribution as well. Filtering is a looping of these two steps every time when new measurements are available.

  By looping the first step, it is possible to get a mean and covariance matrix for every future timestep for the vehicle state. This can be modified into a trajectory mean with linked uncertainty (i.e. normal distribution in each timestep), as showed in [31, 29]. As compared to the approached of *single trajectory simulation*, Gaussian Noise simulation techniques have the benefit of uncertainty representation on the predicted trajectory or movements. However, there are some limitaitons as well: modelling uncertainties employing normal distribution is not quite enough to show the different possible maneuvers. A possible solution for this could be uncertainty representation using Variational Gaussian Mixture Model (VGMM). Author of [32] used Switching Kalman Filter (SKF) for this exact purpose. [30] depends on mass of KF to show possible models for movement evolution for vehicle and be able to freely change between them. [28] introduced an alternative approach: to use heuristics and change different kinematic model depending on the current situation.

- **Monte Carlo Simulation.** In generic case when no assumptions in advance are made about models linearity or uncertainty model, distribution expresion on predicted vehicle states are not clear. Monte Carlo method is the right tool for this kind of situation. The idea under the Monte Carlo method is to randomly sample the input of the dynamic or kinematic model and to generate potential future trajectories. If the road topology is taken into account, various mass can be added to the generated trajectories and movements to penalize the ones which do not respect the restriction of the road design. Kinematic and dynamic models can be used for Monte Carlo method by categorizing inputs instead of considering them as a constant. Typical inputs are categorized to acceleration, steering angle or lateral deviation. To be able to take into account eligibility of the movement, generated trajectory samples, which has a bigger acceleration than physically is allowed can be removed, as it was done in [33] or consider limitations which vehicle has (weight, length, etc.) and distribute dynamic and kinematic models in a more realistic manner and remove all impracticable trajectories from predefined trajectories list as it was done in [34]. Monte Carlo method can be used to foresee trajectory or movements for a vehicle with a very well known current state or for vehicle which has uncertainty in the current state, which was estimated by one of the filtering algorithms.

### 2.2.2 Movement Prediction Using Maneuver-Based Models

*Maneuver-based* motion models show vehicles as independent moving entities, i.e. it is assumed that the movement of a vehicle on the road match to a series of independently executed movements from the other vehicles on the same road. Oxford dictionary [35] a movement/maneuver as "a physical movement or series of moves requiring skill and care". Term

behaviour in literature often is used meaning the same meaning, e.g. in [36, 37, 38], for the sake of simplicity word "movement" or "maneuver" will be used in this work with defined meaning. Movement and trajectory prediction using maneuver-based motion models work with in advance recognized movements which driver possibly intend to perform. If an algorithm can recognize intended movement, the algorithm can assume that future actions of the driver will match the recognized movement. Due to this a *priori* information, trajectories received with this method are more relevant and reliable than the ones received using physics-based motion models. Maneuver-based motion models rely on prototype trajectories or on movement intention estimation.
Vehicle motion classification into maneuver/movement classes has been extremely widely applied not only in driver assistance systems but into natural driving studies [39, 40, 41, 42, 43, 44, 45, 46, 47, 48]. Authors of the majority of approaches are using heuristinc [42] or training classifiers like SVMs in [43], HMMs [39, 44, 45]. Long Short Term Memorys (LSTMs) in [46], Bayesian networks [47], etc., as movement-based features using speed, deceleration, acceleration, yaw rate, lane position, turn signals, distance from other vehicle and other road context information. Authors of [42] classified vehicle's movement into class "keep lane" or "change lane" grounded on how far the closest car is and predicted future trajectory by applying quintic polynomial of the current car movement state and pre-defined ultimate movement state for each movement class, defined before.Authors of [47] used six different movement classes, which were defined before and using DBN based on multiple movements and context based features selected the potentially right future movement. Authors of [48] defined an individual Gaussian process for three movement classes and established a multi-modal distribution for possible future trajectories using each model. However, in the study, only one case-based prediction has been introduced. Authors of [39] also determined separate Gaussian processes, this time for four different movement classes, which were classified using a hierarchical HMM. This method was tested on real highway data. Authors of another study [40] used a random forest classifier for movements classification into pre-defined movement classes: left or right lane changes or keep lane. Authors used a separate Gaussian Mixture Regression (GMR) model for predicting lateral movement for vehicles using each class. Method was tested on real highway data. Similar method, but without predifined movements classes for prediction longitudinal motion for vehicles were used in [41].

### 2.2.3 Movement Prediction Using Intention Aware Models

*Interaction-aware* motion models introduce cars as manoeuvring items which co-operate with each other, i.e. a movement of a vehicle is considered to be affected by a movement of the other moving object in the traffic scene. Keeping into account the dependencies between the separate moving objects leads to a much better explanation of their movement compared with **maneuver-based** motion models described in the previous subsection. As a result, it gives a better perception of the current situation.
Despite this, a relatively small amount of researches is done considering inter-moving-objects interaction in movement prediction. Authors of [49] assigned two movement classes for vehicles approaching an intersection together, applying a polynomial classifier which "punishes" cases that potentially would lead to near-collisions situations. Authors of [50] worked with a much complex scenario and assigned movement classes to multiple together interacting vehicles in a highway scenario. However, foreseen movements, trajectories of a vehicle are assumed to be given in advance. Results reported using a simulated environment. [25] in their work considered multiple interacting vehicles together with the difficulty of estimating their future motion. Authors of [40] not directly used inter-moving-objects interaction by including comparative positions and velocities of vehicles close by as features for movement and trajectory prediction.

### 2.2.4 Movement Prediction Using Data-Driven Model

As mentioned earlier *data-driven* movement prediction can be generally classified into clustering-based and probabilistic approaches. **Clustering-based** approaches group the training data in order to provide a set of possible prototype trajectories [51, 52]. Partially observed trajectories are checked and compared with a prototype trajectory using various distance measurements, as Dynamic Time Warping (DTW), Longest Common Subsequence (LCSS), Hausdorff distance, etc. and after matching movement trajectory with prototype trajectory, later one is used as a model for future movement. Clustering approach is quite easy, but the main disadvantage of this method is the deterministic nature of the predictions. **Probabilistic** approach contrary learn probability distribution of every movement trajectory class and gives the conditional distribution for future movements, given current trajectory. This lets us avoid some degree of natural uncertainty of predicting the future.
Authors of [53, 39] for modelling trajectories and for motion prediction use Gaussian Processes which are the most popular approaches solving prediction problems so far. [40] uses GMR for prediction longitudinal movement of a vehicle, while [41] uses the same method for lateral movement prediction. [54] uses VGMMs for conditional distribution within snippets of future having snippets of movement history models. The latest approach is much easier and computationally more effective when compared to Gaussian Process Regression. Authors proved the efficiency of method predicting non-linear movements in turns at the intersection scenarios.

### 2.2.5 Limitations of Methods for Movement Prediction

Subsections 2.2.1, 2.2.2, 2.2.3 and 2.2.4 described movement prediction with different feature based model. This subsection will introduce limitations of all these methods.

- **Physics-based approach.** Predictions using physics-based motion models are restricted to very short-term ( $< 1$ sec.) motion prediction due to low-level motion (dynamic and kinematic) properties this method relies on. Usually using this method it is unable to foresee any change in the vehicle movement which happens due to an execution of a particular maneuver (e.g. speed up, slow down, make a turn, etc.), or changes caused by external factors (e.g. slowing down due to traffic lights, signs, other vehicles, etc).

- **Maneuver-based approach.** For a very long time, the biggest limitation of prototype trajectories was time representation. When the movement models are showed using a finite set of trajectories it takes a very large number of prototypes to represent the large variation in the implementation of an every possible movement pattern. Handling subtle situation in traffic, as movements with waiting time at a stop line, not constant velocity caused by traffic is a very big issue for such models. For a certain extent, Gaussian Processes (GP) were introduced. They solved this kind of problem by introducing time-independent movement patterns [53]. On the other hand, GPs have some other limitations as well. First of all to be able to take into account all possible traffic scenarios, has very heavy computational time, despite that they are not considering the physical limitations of a vehicle and due to that may generate or predict unrealistic trajectories and movements. To solve these problems the best solution so far was proposed in [55]. Authors used Rapidly-exploring Random Tree (RRT) to be abe to "randomly sample points toward dynamically feasible trajectories, using as inputs the current state of the vehicle and the sample trajectories generated by the GPs" [55]. Another issue with using predefined prototype trajectories is an adaptation to a different road, i.e. for different intersections. Each movement model is defined for a specific road/intersection geometry and topology, what means that prototype models only can be used with the same or very similar topology.

  Maneuver-based approach contains similar limitations which described under limitations of data-driven approach.

- **Interaction-aware approach.** Prediction using interaction-aware motion models are the most exhaustive method suggested in the literature so far. Using it, is possible to predict for a longer-term as compared to physics-based motion prediction models, and predictions are more trustworthy than using maneuver-based motion models in predictions due to taking dependencies between the surrounding cars into consideration. However this completeness has some disadvantages as well: calculation of all possible trajectories with all possible models take a lot of time and because of that, it is not very compatible with using in real-time situations. For this reason, using interaction-aware motion predictions are not so popular.

- **Data-driven approach.** The assumption that movement of vehicles do not depend on each other and other traffic participants is not accurate. All vehicles without any exceptions use a road together with other traffic participants and movement performed by one vehicle directly or indirectly effects others. Dependencies with each other are quite strong at intersection, where road rules, not only movement of other cars must be taken into account. Ignoring these reliances can lead to wrong interpretations of the situations, and affects the evaluation of the risk. A data-driven approach is quite easy, but not always it pays attention to these critical dependabilities and it is quite difficult to pre-define all possible action of other traffic participants, i.e. the main disadvantage of this method is the deterministic nature of the predictions.

# 3 Probabilistic Trajectory Prediction

This chapter introduces a concept and theoretical framework for all steps which are necessary to solve the problem of motion recognition and prediction with our proposed probabilistic intention prediction algorithm.

## 3.1 Including Prior Information To Algorithm

For testing the probabilistic intention prediction algorithm we have two environment setups: X and T intersection, Figure 3.1. Information about the map is crucial because it gives the value of initial belief ($b_0$), one of the most important variables of our initial probabilistic model at the beginning of prediction-making.

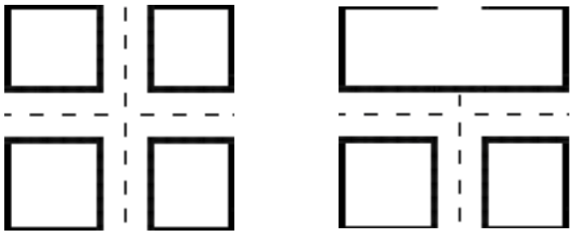

**Figure 3.1:** X intersection map in the left and T intersection map on the right

In the map with X intersection, we assume that the probability to go to any direction is equal, so initial belief value is $b_0[\text{right, straight, left}] = (0.333, 0.333, 0.333)$. In the case of T intersection from the very beginning, we know that one direction is not available - the car can go only straight or right. Considering this information we need to recalculate the value of initial belief since it can't be equal anymore. Initial belief on T intersection is calculated using information about number of demonstrations we used for algorithm learning probabilistic prediction model. We calculate initial belief for right and left directions using these formulas:

$$b_0(right) = \frac{R}{L+R}, b_0(left) = \frac{L}{L+R}, \tag{3.1}$$

where R and L stand for trajectories going to the right and left in our predefined Database (DB). To be able to define initial belief, the first step of intention prediction algorithm is to recognize the map. There were two map recognition methods tested in the algorithm. Results proved that both methods for map recognition are working, but the first method is a very simple one and used only contours in the map, which means that is a map will have slight changes like new obstacles, new code adaptation would be needed with this method. Due to this reason another map recognition program was created which uses SSIM and recognize map according to some common similarities - all description is in the previous chapter.
Subsections below will give an explanation of two different methods, both were using Open Source Computer Vision Library (OpenCV) library for Python, we used for map recognition. OpenCV - is an open source library for computer vision and machine learning programs. It was developed to grant a common infrastructure for computer vision applications and to speed up the usage of machine apprehension [56].

### 3.1.1 Map Recognition using Contours

Map recognition using contours in OpenCV library is a relatively simple method, but a helpful tool for object recognition (as well as, for object detection or/and shape analysis), using the number of counters in the pictures and according to it, making recognition.
First of all, let's answer the question, what are contours? We can define contours as a curve which joins all continuous points, which has the same colour or the same intensity. The tool works in this way, that before contours detection it applies threshold (as in our case) or canny edge filter. Contours in OpenCV works in greyscale and to have the best results object of interest should be white in black background (or vice versus) [57]. Let's use the code snippet to explain contours better:

```
import numpy as np
import cv2

img1 = cv2.imread('maps/testmap.png')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
ret1, thresh1 = cv2.threshold(gray1, 127, 255, 1)
contours1, h1 = cv2.findContours(thresh1, 1, 2)
```

In cv2.findContours() function there are 3 arguments: first of them is source image, converted to greyscale. The second one is contour retrieval mode (we assigned it to be 1 since we are interested to retrieve contours only, not to create any hierarchy/parent-child level). The last argument means contour approximation method. Contour approximation modified a contour shape to another shape with fewer peaks depending on the accuracy we choose (using Douglas-Peucker algorithm [58]).
To understand this, let's use another snippet of code:

```
approx1 = cv2.approxPolyDP(cnt1, 0.01 * cv2.arcLength(cnt1, True), True)
```

Without function which approximate shape we usually are not getting the correct form, due to some "visual information noises" or due to other problems in the source image. Using shape approximate function we use maximum distance from contour to approximated contour - this is a parameter of precision. This parameter is the second argument in approxPolyDP() function ("0.01 * cv2.arcLength(cnt1, True)"). To chose this parameter correctly is important because due to it we receive the good or bad result of finding the contour method.
To see the output of the method we can draw contours, using cv.drawContours function (code snippet below):

```
cv2.drawContours(img1, [cnt1], 0, (0, 0, 255), -1)
```

The first argument of the function is the source image, the second argument are contours which were found, the third argument is counter index and the last argument is colour. The Figure 3.2 shows how the output of function looks like.

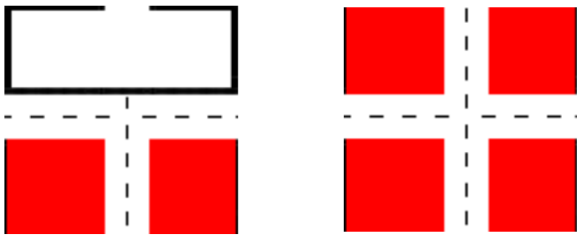

**Figure 3.2:** Contours for T (left) and for X (right) intersection maps, received as one of outputs of map recognition, using contours method

Having this information map is recognized by a number of contours it has in the image source (T intersection has 4 contours and X intersection has 8). As mentioned before this method is relatively simple to use, but it will cause problems in map recognition if there will be more contours inside of the image of the map (i.e. if there will be some obstacles depicted on the map T intersection) and it will cause problems in recognition. Due to this reason another map recognition method was proposed.

### 3.1.2 Map Recognition using SSIM

For humans, it is quite easy to find differences between images, we are quick to recognize what is missing in one or another image - we are good at noticing how much one image similar to another. The first method we described for map recognition cannot tell how much images differs between each other but for better map recognition we need to be able to recognize similar forms in the picture (e.g. we should be able to recognize T intersection even if one image of T intersection has obstacles on the road or trees, houses on the side, etc.). For another type of map recognition, we proposed to use SSIM. SSIM is a method for estimating similarities between two images. The SSIM index can be described as a measure of quality for one image after comparing with another. Mode details on SSIM can be found in [59].
To be able to work with SSIM at first we need to have any size DB of images of the same object (set of X and T intersection images). Figure 3.3 show dataset for testing images similarities for T intersection.

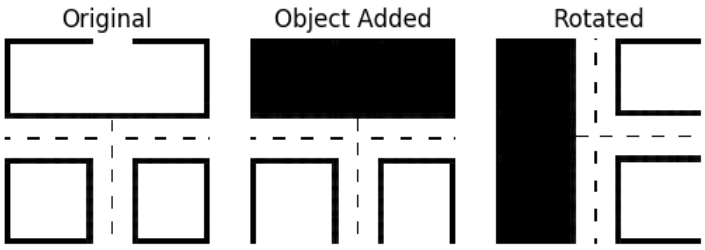


**Figure 3.3:** Dataset for T intersection recognition, used for testing SSIM method

For image similarity SSIM method calculates a Structural Similarity Index by the formula below:

$$SSIM(x,y) = \frac{(2I_xI_y + Cont_1)(2SCont_{xy} + Cont_2)}{(I_x^2 + I_y^2 + Cont_1)(SCont_x^2 + SCont_y^2 + Cont_2)} \tag{3.2}$$

where Cont is contrast and I and SCont are estimated intensity and estimated signal contrast respectively.
SSIM tries to model the comprehended difference in the structure of the image. During calculation of the structural similarity index, two windows (i.e. small sub-samples) are compared. This allows us to reach robust approach that is able to see changes in the image structure, rather than just the perceived change. Structural similarity index formula considers position (x and y values) of the N x N window in each image, the mean of the pixel intensities and the variance of intensities in both directions, along with the covariance values [60]. Figure 3.4 shows structural similarity index value for different images from data set as compared with original one.

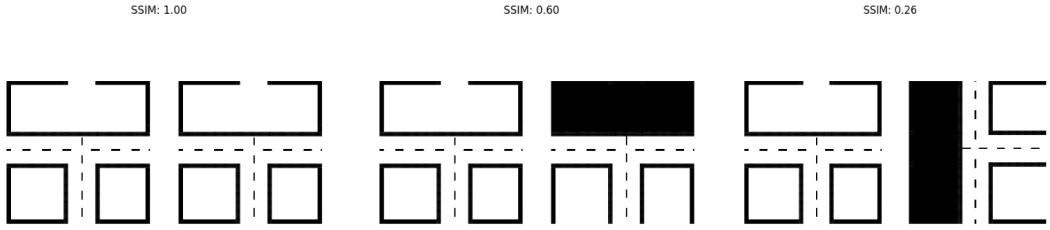


**Figure 3.4:** SSIM value for different images from data set as compared with the original one. Index of similarity for the original picture is equal to 1, for pictures with added object similarity index is equal to 0.6 and index of similarity for a rotated picture with object added is 0.26. SSIM index can vary from -1 to 1

SSIM value is between $-1$ and 1, where $-1$ shows no similarity and 1 shows perfect matching. We can choose the level is similarity (value of structural similarity index) is good for our recognition.

## 3.2 Trajectory Preprocessing

Our DB has a various number of demonstrations of different movement classes, which we used for teaching algorithm initial probabilistic prediction model. Since all trajectories are different movement wise, they are not equal by length (i.e. one trajectory might have $8,000$ time steps, another one $7,600$, third $9,621$, etc.). This happened because of the different pace of car while recording data, different movements, like turning, which can reduce or improve the length of the trajectory. This is a problem for probabilistic intention prediction algorithm. Having different lengths trajectories makes some problems in further calculations, comparison of results, the calculation over thousands of steps also has a longer computation time, etc. Because of that, trajectories unification is necessary (at least with trajectories we already have recorded, for trajectories which are getting in real time additional step is necessary and it is described at the end of this chapter).
Trajectories unification was done using interpolation when all trajectories transformed to have an equal number of time steps (chosen by us), in spite of the fact that original number of time steps are unequal. In mathematics interpolation is a method of building new data points inside the range of a discrete set of known/given data points.
If we consider that we have point A and point B with coordinates $(x_A, y_A)$ and $(x_B, y_B)$, formula for finding values for interpolated point with coordinates $(x_{int.}, y_{int.})$, can be calculated using formula below:

$$y_{int.} = y_A + (y_B - y_A)\frac{x_{int.} - x_A}{x_B - x_A} \tag{3.3}$$

The pseudo code below (Figure 3.5) shows the way of interpolation in this project.

```
1  begin
       x, y        # coordinates of trajectory which needs to be interpolated

       # Calculate the n-th discrete difference along the given x and y axis
2      xd[n] = a[n+1] - a[n]
3      yd[n] = a[n+1] - a[n]

       # Calculate Euclidean distance between xd[n] and yd[n]
4      dist = np.sqrt(xd[n] ** 2 + yd[n] ** 2)

       # Calculate cumulative sum of the elements along dist
5      u = np.cumsum(dist)

       # Stack array u in sequence column wise (horizontally)
6      u = np.hstack([[0], u])

       # Calculate evenly spaced numbers over a specified interval [start, stop]
7      t = np.linspace(0, u.max(), len_des)

       # Calculate the one-dimensional linear interpolation with given discrete data points
8      xn = np.interp(t, u, x)
9      yn = np.interp(t, u, y)
10 end
11 return xn, yn # interpolated coordinated of given trajectory
```

**Figure 3.5:** Pseudo code for interpolation process

Figure 3.6 illustrates 3 original trajectories for each movement class in red and 3 interpolated version of the same trajectories in blue. Original trajectory, which goes to the right has 7,581 time steps, while straight trajectory has 13,666 and the left trajectory has 10,929 time steps after interpolation all trajectories contain 10 steps.

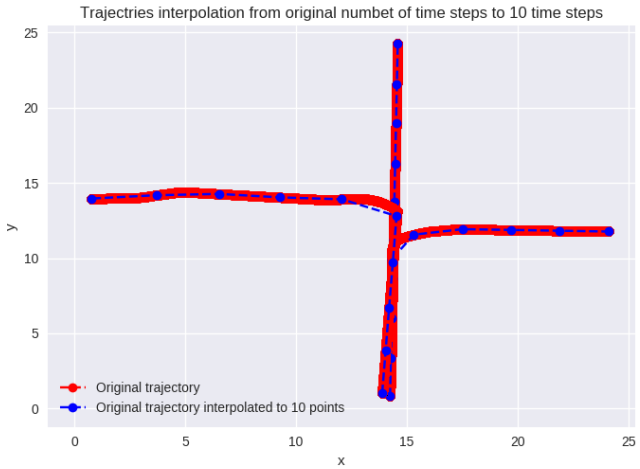


**Figure 3.6:** Original and Interpolated Trajectories. Red lines show the original trajectories of going to the right, left and straight direction. The number of time steps which has original trajectories which go to the right, straight and left have 7,581, 13,666 and 10,929 time steps respectively. Blue dotted line represents interpolation of the same trajectories. Interpolation was done for 10 time steps

As can be seen, interpolation does not distort trajectories and can be used in further calculations.
After all trajectory unification is done, mean of trajectories and variance per each time steps need to be found. More

information how to do this is in this chapter below.
After we include unification process for demonstrations into our algorithm, learning of probabilistic model begins.

## 3.3 Learning Probabilistic Trajectory Models From Demonstrations

Our proposed probabilistic intention prediction algorithm learning from the set of previously recorded demonstrations. More about how data was collected is written in Chapter 4. Every trajectory can be considered as a generic vector, consisting of all values learned that the time period and we also have a vector of all coordinates, which was recorded at a particular time step $t$. Trajectories can be mono-dimensional or multidimensional, in our case we are working with two-dimensional space, having x and y coordinates. We want to note that due to different trajectories, the length of each trajectory is not the same. To find a common representation for learning purposes, we needed to have the same length for all trajectories. We applied interpolation to achieved that (more details in this Chapter, just above).
To check every trajectory in DB is computationally difficult, takes much more time to process and give not so much movement freedom while testing algorithm. To avoid that the mean trajectory and covariance matrix for each movement class, using formulas 3.4, 3.5:

$$\mu_x = \frac{\sum_{i=1}^{n} x_i}{n}, \mu_y = \frac{\sum_{i=1}^{n} y_i}{n} \tag{3.4}$$

$$\mathbf{\Sigma x} = \frac{\sum_{i=1}^{n} (x_i - \mu_x)(x_i - \mu_x)^T}{n-1}, \mathbf{\Sigma y} = \frac{\sum_{i=1}^{n} (y_i - \mu_y)(y_i - \mu_y)^T}{n-1}, \tag{3.5}$$

where $x_i$and$y_i$, x and y values from all trajectories at time step $t$, n is number of trajectories, $\mu x$ and $\mu y$ are mean values of x and y from all trajectories at time step, $\Sigma x$ and $\Sigma y$ covariance matrix.
The proposed algorithm uses the distribution over the observed trajectories. To recognize movement class, we use the movement that passes by the mean of the distribution.
When we covered all this and we have probabilistic prediction model learned we can go into an explanation of probabilistic estimation methods.

## 3.4 Probabilistic Estimation Methods for Movement and Intention Prediction

Reasonable prediction and following decision-making process require considering uncertainty and objectives for the current situation. In this section, uncertainty will be represented as a probability distribution.
Uncertainty can be a result of partial information about the state of the world. In a real world trying to fulfill any given task, it is possible to meet various reasons which do not allow to finish a task without any difficulties. This means, that with information we have at hand, it is hardly possible to make a task evaluation with complete certainty.
Uncertainty can appear from practical and theoretical limitations while trying to predict future events, e.g., trying to exactly predict how a human would react in one situation or another, a decision support system would need to consider a model of the human brain. Even if the operation is known very well, it is still difficult to predict the end state and next actions which will be taken, due to spontaneous failures or other agent actions.
A robust prediction (and later decision) making system need to take into account sources of uncertainty, which exist in the current state and consider it when computing the future outcomes for events. In order to describe uncertainty computationally, it needs to have a formal representation.

### 3.4.1 Belief State and Probability

Solving tasks which involve uncertainty, it is very important to be able to compare the credibility of different statements. For example, if belief for action E is stronger than our belief for action T, then E $\succ$ T. If E and T have the same degree of belief, then E $\sim$ T.
It is also beneficial to be able to compare beliefs about statements considering some given information, e.g., we can say that likelihood that action C may happen while E condition is happening is bigger than having T, then this expression would be written (E | C ) $\succ$ (T | C ).
In order to make particular assumptions about the relationships of the operators $\succ$, $\prec$ and $\sim$. The assumption of *universal comparability* and *transitivity* assumptions requires to hold the same mathematical rules. Both assumptions allow representing degrees of belief by a real-valued function [61], i.e. probability function P can be expressed like that:

$$P(A|C) > P(B|C) \iff (A|C) \succ (B|C)$$

$$P(A|C) = P(B|C) \iff (A|C) \sim (B|C),$$

where $P(A|C)$ and $P(B|C)$ are conditional probabilities, with A, B and C events. $P(A|C)$ can be read as "the conditional probability that even A will happened given some event C". If new assumptions about the probability P form, then P need to satisfy the main axioms of probability: $0 \leqslant$ P (A | B ) $\leqslant 1$. If we are sure that A action will happen when B action is given then P (A | B ) = 1. If A action will not happen when B action is given, then P (A | B ) = 0.
Deep review about probability theory won't be provided in here, but this work relies on important probabilities properties. The first of them is a definition of *conditional probability*:

$$P(A|B) = \frac{P(A,B)}{P(B)}, \tag{3.6}$$

where P(A, B) shows the probability of A and B both being true.
Another property which is important is the *law of total probability*, which states that if $\beta$ is a set of "mutually exclusive and exhaustive propositions" [61], then

$$P(A|C) = \sum_{B \epsilon \beta} P(A|B,C)P(B|C) \tag{3.7}$$

Finally, the most important rule for further work comes from the definition of *conditional probability*:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{3.8}$$

This equation is known as ***Bayes' rule***, where $P(A|B)$ is **posterior**, in other words, it is something we are trying to evaluate, $P(B|A)$ is is likelihood, it is the probability of observing the proofs, given having an initial hypothesis, $P(A)$ is a **prior**, it is the probability of hypothesis without any additional information from before and $P(B)$ is a **marginal likelihood**, it is the complete probability of observing evidence for our hypothesis. This formula will be very important for the following work.
But still, after this short introduction, question what exactly belief state is, still exists. One option to answer this would be the most believable next state for an examined object, considering experience in the past, which is given. This idea can be sound basis for predictions in some cases, but in general, this idea is not sufficient. Being able to operate efficiently the degree of uncertainty must be taken into account, e.g. if the main agent is confused what future state could be, it could be good to ask directions, take a look into the map, search for reference point, etc.
Other options for belief computation would be using probability distributions over states of the world, which we have. In this case, distributions encode the subjective probability for the main agent and include information about the state of the world and give a basis for taking action under uncertainty we have. Moreover, sufficient statistical information of action made in the past and initial belief state of the agent is comprised, i.e. computed belief state for the current agent's state and additional information about its past observations and/or action made, would provide any further information about the current state of the world [62].
**Computing belief states**[62]:
A belief $b$ is a probability distribution over state space $\mathcal{S}$, *b(s)* is the probability set to world state $s$ by belief state $b$. The axioms for belief state is the same as for probabilities: $0 \leqslant$ b(s) $\leqslant 1$, for all s $\epsilon$ $\mathcal{S}$ and $\sum_{s \epsilon \mathcal{S}} b(s) = 1$. At every new step, new belief *b'* must be computed given old belief $b$, an action $a$ and an observation $o$. The new belief of an new state *b'(s')* can be calculated using formula:

$$\begin{aligned} b'(s') =& Pr(s'|o,a,b) \\ =& \frac{Pr(o|s',a,b)Pr(s'|a,b)}{Pr(o|a,b)} \\ =& \frac{Pr(o|s',a)\sum_{s \epsilon \mathcal{S}} Pr(s'|a,b,s)Pr(s|a,b)}{Pr(o|a,b)} \\ =& \frac{O(s',a,o)\sum_{s \epsilon \mathcal{S}} T(s,a,s')b(s)}{Pr(o|a,b)}, \end{aligned} \tag{3.9}$$

where the denominator of formula 3.9, Pr(o | a, b), can be interpreted as a normalizing factor, which is independent of next state s', which causes the sum of belief of all possible next states to 1. The task of the state estimation function SE(b, a, o) is to update the belief state based on the *a, o* and the previous $b$, as its output gives new belief for new state *b'*, $T(s,a,s')$ is a transition from one state to another probability function.
In later work, belief update will act an important role, but it will be computed using different components. Further detail description of belief computation related to this work will be provided.

### 3.4.2 Modeling Belief for Prediction Making

A Bayesian version of a Gaussian Mixture Model is used for calculating the belief each trajectory class and this section gives definitions for beliefs representing for a car over trajectory movement classes (right,straight and left).
Our observation is the current position of the car, which means that trajectory class is partially observable, due to that our belief is represented as a probability distribution over all trajectory classes.
For maintaining correct belief, updates must be done every time step. Updating belief constantly is important because sudden position change can show drivers' intentions and what his next steps could be. To be able to predict possible future movement, current position information need to consider every time when belief update is calculated. The belief update is calculated using Bayes Rule, formula 3.10:

$$\begin{aligned} b_{t+1}(k) &= Pr(k|o_t, b) \\ &= \frac{Pr(o_t|k, b)Pr(k|b)}{Pr(o_t|b)} \\ &= \frac{Pr(o_t|k, b)Pr(k|b)}{\sum_j Pr(o_t|b)Pr(j|b)} \\ &= \frac{Pr(o_t|k, b)b_t(k)}{\sum_j Pr(o_t|j, b)Pr(j|b)} \end{aligned} \tag{3.10}$$

With given formula belief for future step $b_{t+1}(k)$ for a predefined class is calculated. $b_t(k)$ is a belief from the last calculation, denominator of function is a normalization function for the current step. $Pr(o_t|k, b)$ is the car position observation model, which returns the probability (or likelihood) of going to any direction from the current position, observed with $o_t$. This likelihood can be calculated using multivariable Gaussian probability distribution function $\mathcal{N}(x|\mu, \Sigma)$. This probability distribution function looks the following:

$$\mathcal{N}(x|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n det(\Sigma)}} exp(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)), \tag{3.11}$$

where dimensionality n = 2 and x is a currently observed position of the car $x = (o_t^x, o_t^y)^T$, $\mu$ is a mean from all predefined trajectories in DB for the same class $\mu_k = (\mu_{t,k}^x, \mu_{t,k}^y)^T$ and $\Sigma$ is a covariance matrix of all predefined trajectories in data base for the same class $\Sigma_k = (\Sigma_{t,k}^x, \Sigma_{t,k}^y)^T$. Mean for x and for y coordinates and variance was found before using formulas 3.4 and 3.5

The denominator of formula 3.10 is the so-called normalization factor, which sums all likelihoods of the car over all movement classes at the current time step. The final result of formula will return updated belief that car is moving towards any of classes.
Figure 3.7 shows pseudo-code how belief updates are made over time, as an input having current belief $b_t$ at time step t, current car position $x_t, y_t$ from the last made observation, as well we have before calculated mean and covariance values at that time, $\mu_{t,k}^x, \mu_{t,k}^y$ and $\Sigma_{t,k}^x, \Sigma_{t,k}^y$ respectively.

```
1  begin
     b_t          #current belief
     x_t, y_t     #observed current car position
     Σ_t          #mean (x and y values) of predefined trajectories in current time step
     μ_t          #covariance matrix (2x2 size) of predefined trajectories in current time step
     b_t+1 ← Ø    #updated belief (empty set)
     η ← Ø        #normalization factor (empty set)

   #normalization factor calculation for each observation probability
2    for k∈K do
3        p_k = f((x_t, y_t), μ_t, Σ_t) from N((x_t, y_t) | μ_t, Σ_t)
4        η = η + p_k
5    end

   #new goal distribution over classes calculation
6    for k∈K do
7    b_t+1(k) ← p_k b_t(k) / η
8    end
9  end
10 return b_t+1
```

**Figure 3.7:** Pseudo Code for Updating belief

This cycle repeats while the car is moving and at every time step, it gives a new prediction of what future movement is going to be. Results of testing probabilistic intention prediction algorithm are given in the next chapter.

## 3.5 Trajectory Scaling

Computation time for prediction is one of the main things which needs to be fast. Even though more belief updates give more precise results, more computations take more time. In this project, we took various numbers of time steps and compared results to get them as precise as possible and have fewer time steps (e.g. we aimed to keep precision of trajectory of 100-time steps but have only 10-time steps trajectory).
To achieve desired results, we used "toy problem" approach. [63] defines it as "In scientific disciplines, a toy problem is a problem that is not of immediate scientific interest, yet is used as an expository device to illustrate a trait that may be shared by other, more complicated, instances of the problem, or as a way to explain a particular, more general, problem-solving technique".
We raised a likelihood (formula 3.11) of the trajectory with a bigger number of time steps at every time step $\frac{1}{\frac{T}{t}}$ and updated formula 3.11 looks like shown below. After scaling we compared results of matching points in each trajectories. After these modifications formula 3.10, looks like:

$$b_{t+1}(k) = \frac{(Pr(o_t|k,b))^{t/T} b_t(k)}{\sum_j (Pr(o_t|j,b))^{t/T} Pr(j|b)}, \tag{3.12}$$

where T and t are the bigger and the smaller numbers of times step in compared trajectories. To simplify all this, let's have an example: let's imagine we have the trajectory interpolated to 100-time steps and we want to see how results differ when we do belief updates all 100 times with doing belief at every 10th step (10th, 20th, ..., 90th, 100th). Without doing any changes in the original code, these results are not matching, in fact, they differ quite a lot. But if we raise likelihood (formula 3.11) of the trajectory where belief is updating 100 times by $\frac{1}{\frac{100}{10}} = \frac{1}{10}$ at every time step and then do belief updates as normal and compare them with belief updates which are calculated every 10th step with making no changes in the original code. After comparison of these results, it was easy to see that results are matching or are close enough to each other. The pseudo code of the given example is in Figure 3.8.

```
1  begin
     N          #number of time steps (e.g. 100)
     n          #number of time steps for scaling (e.g. 10)
     b_t        #current belief
     x_t, y_t   #observed current car position
     Σ_t        #mean (x and y values) of predefined trajectories in current time step
     μ_t        #covariance matrix (2x2 size) of predefined trajectories in current time step
     b_t+1 ← Ø  #updated belief (empty set)
     η ← Ø      #normalization factor (empty set)

     #belief  update calculation for N time steps
2    for i in range (1, N+1) do
       #normalization factor calculation for each observation probability

3      for k∈K do
4        p_k = f((x_t, y_t), μ_t, Σ_t)^1/(N/n) from N((x_t, y_t) | μ_t, Σ_t)^1/(N/n)
5        η = η + p_k
6      end

       #new goal distribution over classes calculation
7      for k∈K do
8      b_t+1(N)(k) ← p_k b_t(k) / η
9      end

       #take the every nth step from trajectory with N steps
10     if i % n == 0 do

11 - 17  #Belief update calculated exactly the same as in Figure 4.4

18     end
19 end
20 return b_t+1(N), b_t+1(n)
```

**Figure 3.8:** Pseudo Code for Scaling Trajectory for Belief Update

Having this in mind we can make our prediction making the process faster and more precise.

## 3.6 Making Prediction While Working With Real Time Data

Another simulation experiment was to try to predict movement with data which is not recorded in advance. For that the same probabilistic intention prediction algorithm was used, just observations about current position came not from the trajectory which was prepared (recorded and then interpolated) for calculations in advance, but from a real-time simulation with ROS and its visualization in RViz, meaning that we did not have all trajectory from the beginning, which forced us to slightly change probabilistic intention prediction algorithm. When we were working with predefined testing trajectories, we treated them exactly the same as algorithm training data: we interpolated trajectory with the same amount on time steps as we did with trajectories from DB and then for prediction making, to formula 3.10 we put the current position of the testing trajectory for the first time step, took mean trajectory position and variance value for the first time step from our DB and so on until the trajectory is over. Due to the problem that we have no idea how many steps trajectory will have, we could not interpolate our testing trajectory and we can't treat the nth step in interpolated (dataset) trajectory with nth step from our real-time testing trajectory, because these steps can be very far away from each other and prediction results will make no sense and the other reason that interpolated trajectory will end at some point (much faster than our test drive in simulation) and then we will get an error, because there will be no interpolated values (i.e. our data set is interpolated for 10 steps, the first 10 steps are just beginning for testing trajectory and during 11th step we will get an error since there is no 11th step in interpolated data set and so on).
Because of that, we used the current observed position and using Euclidean distance formula 3.13 to calculate the distance between the current car position and all points in the interpolated trajectory.

$$dist = \sqrt{(x - x_{int.})^2 + (y - y_{int.})^2}, \tag{3.13}$$

where x and y are currently observed position values $x_{int.}$ and $y_{int.}$ is values per each time step of interpolated trajectory. Having all these values calculated, the smallest distance value is selected, Having that to formula 3.10 we can put the closest value of mean trajectory and variance values for the same time step.
There are other existing methods, which could be adopted for find differences between testing and trajectories in DB, such as DTW in [64] or [65] which incorporated local variability in the execution of position.

### 3.6.1 Probabilistic Movement Primitives (ProMPs)

TO FIX
As in previous cases, we have a set of demonstrations which are pre-recorded in advance. We can see each demonstration as vector or time-dependent variables, and each time step as a vector, containing all variables which were learned at that particular timestep $t$. Our demonstrations are two-dimensional (however, they can be mono- or multi-dimensional). As in previous cases, the length of the demonstrations are not matching, due to that, we used the same time modulation for 100-time steps as before.
ProMPs uses a generic Bayesian parametric model to express demonstrations:

$$x_{reconstructed}(t) = \Phi_t \omega, \tag{3.14}$$

where $x(t)$ x dimension depending on time step $\omega \epsilon R^M$ is RBF weights, not depending on time, $\Phi_t$ is a vector of RBF, at time step $t$, parameter meaning the number of RBF is M. $\Phi_(t)$ is expressed as:

$$\Phi_t = [\psi_1(t), \psi_2(t), ..., \psi_M(t)], \tag{3.15}$$

with

$$\left\{ \begin{array}{c} \psi_i(t) = \frac{-(t-c(i))^2}{2h} \\ c(i) = i/M \\ h = 1/M^2 \end{array} \right\}$$

where M denotes the number of RBF, h denotes standard deviation and c denotes mean. For each demonstration we compute one weight parameter vector $\omega_i$, to have expression of formula 3.16 for each RBF. Weights for each RBF are computed using, Least Mean Square algorithm, together with added diagonal term (in case matrices in Least Mean Square algorithm are not invertible). $\omega_i$ for x dimension is calculated using:

$$\omega_i = (\Phi_t^T \Phi_t + \lambda)^{-1} \Phi_t^T, x_i(t), \tag{3.16}$$

where $\lambda$) is a diagonal term, the smallest singular value of matrix $\Phi_t^T \Phi_t$. Once we obtain all values for $\omega = \omega_1, ...., \omega_n$, we calculate Normal distributions:

$$p(\omega) = \mathcal{N}(\mu_\omega, \sum \omega), \tag{3.17}$$

with

$$\mu_\omega = \frac{\sum_{i=1}^n \omega_i}{n}, \sum \omega = \frac{\sum_{i=1}^n (\omega_i - \mu_\omega)^T (\omega_i - \mu_\omega)}{n-1}, \tag{3.18}$$

where $\mu_\omega$ is a mean if weights and $\sum_\omega$ is covariance. Having the distribution over demonstrations it is possible to represent movement primitives which pass through the mean value. This representation can be found in Chapter 5, Section 5.1.4.
But before representing trajectory with leaned movement primitives, it is very important to choose the right number for RBFs. To few functions can lead us to insufficiency of trajectories approximation, while too many can cause over-fitting problems. We chose the right number for RBFs by computing root mean square deviation, which can be computed as follows:

$$\epsilon = \frac{\sum_n (\xi - \xi_{reconst})^2}{N}, \tag{3.19}$$

where $\epsilon$ is error for trajectory, N - number of trajectories, $\xi$ - original trajectory and $\xi_{reconst}$ - reconstructed trajectory.


**Figure 3.9:** Error value between original and reconstructed trajectories

As it is possible to see from Figure 3.9, having 6 and more RBFs gives the smallest error. Since more RBFs, means the longer computation time, from now we will use 6 and a constant number for RBFs, while working with this set of demonstrations. Figure 3.10

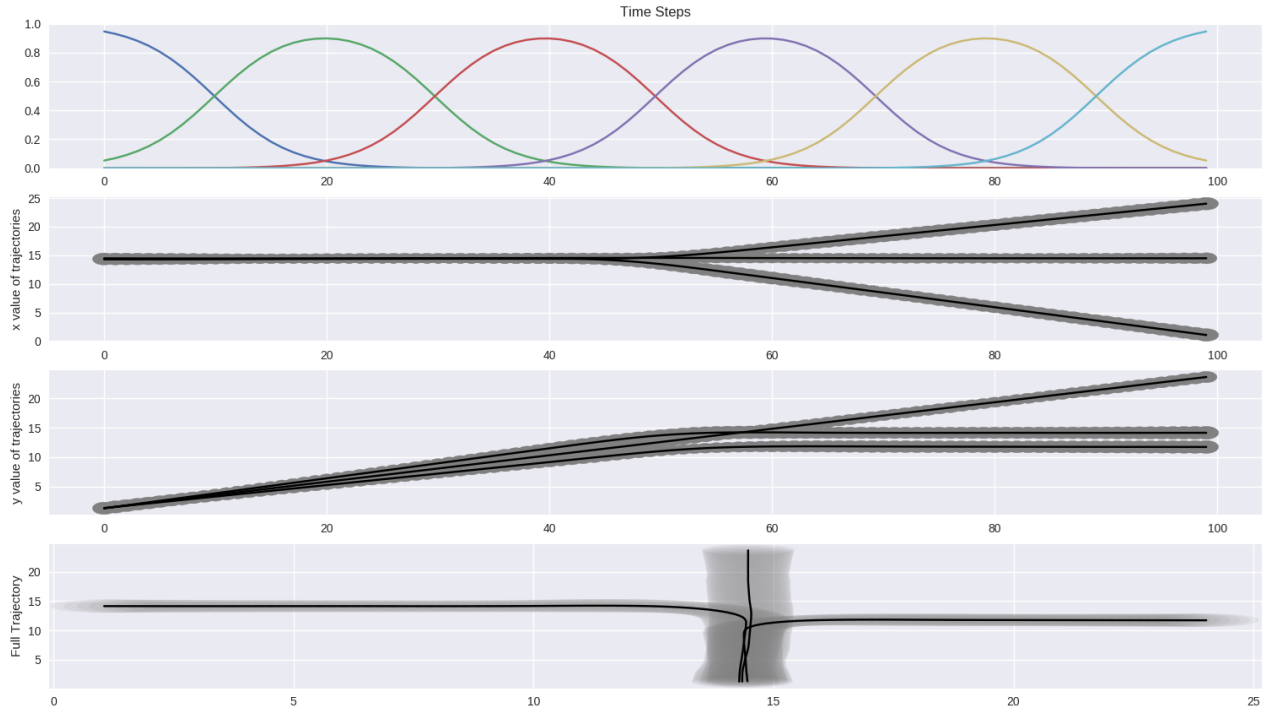


**Figure 3.10:** All primitives from ProMPs. The first picture shows how RBFs looks like, the second and third pictures shows primitives and demonstrations for x and y dimensions respectively and the last picture shows the full trajectory

# 4 Setup and Implementation

This chapter is focused on the implementation of the probabilistic intention prediction algorithm.
The algorithm was written in Python and C++ programming languages and tested, ROS. As mentioned in Chapter 1, ROS is a framework where testing cases can be easily developed and tested. In current visualization framework, we tested 2 environments consisting of a static map for T and X intersections and the car itself (model is created with two cars which can be simulated simultaneously, but for proving the concept we decided to work with only one car at the time).
In order to visualize simulation RViz, which is a 3D visualizer for showing data and state information from ROS, was used. A snapshot of the simulation environment is in Figure 4.1.

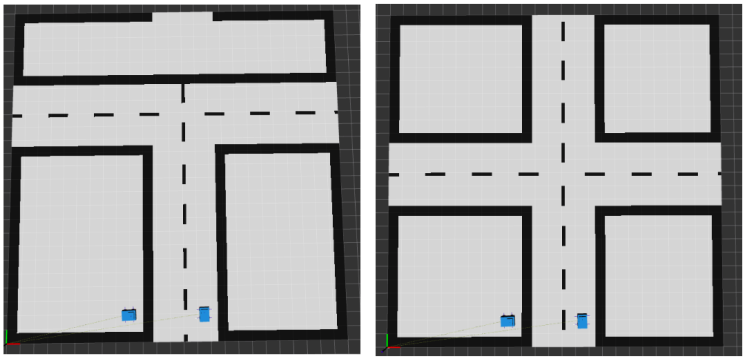

**Figure 4.1:** Experiments were made having two environments: T and X intersection. The figure illustrates how these environments look like in RViz - ROS visualization tool, where simulations were run

As mentioned in Chapter 3, this project has several testing cases. Methods tested with having two types of trajectories. The first one was pre-recorded trajectories and the second one was real time trajectories. Data from every run is stored in order to analyze and deduce what effect each method has on algorithm performance. Parameters and results for the run are independent and tested several times in a specific environment to be able to draw conclusions from the results we received during the testing phase.

## 4.1 Data Collection

Collecting data was one of the first steps for implementing a probabilistic intention prediction algorithm because, in order to recognize and predict intentions, common features for each movement classes need to be described (algorithm has to learn the features from recorded demonstrations in training process). We did not use any existing DBs - we created DBs for trajectories of interests by ourselves.
At the beginning, we collected data only from ROS simulations and later one we collected data from a real car.

### 4.1.1 Data Collection with ROS

To be able to control car in ROS and its visualization in RViz environment, ROS joy package was used. This package allows us to connect a physical joystick to ROS and provides all necessary drivers and libraries. To be able to record and store data which we collected by making experiments Python GUI application was used. On the left of Figure 4.2 brief scheme of data collection is shown and on the right side GUI application window is illustrated.

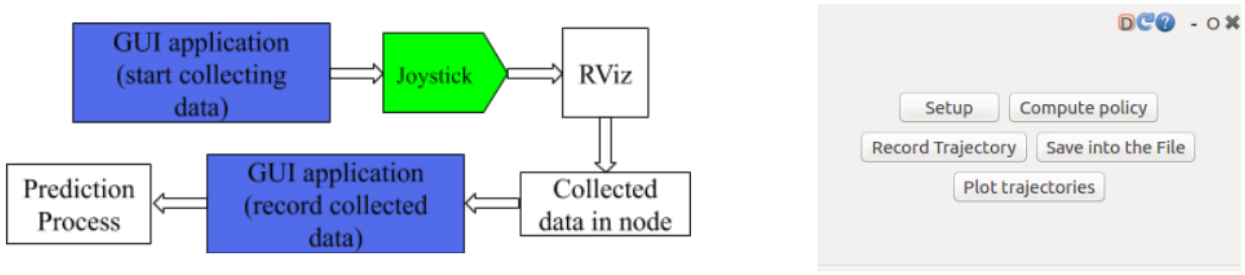


**Figure 4.2:** On the right side graphical user interface is shown, it helps with the simulated data recording process, left side briefly illustrates data collection process, using ROS Joystick node

Joy package is a ROS driver for a "generic Linux joystick".The package consists of joy_node, a node which allows communication between a generic joystick and ROS. Joy message, which contains information about the current state of each button on joystick is published by node [66], having this information command start to move, turn to any direction, stop is sent to the object we want to control in ROS environment.
When all necessary systems were connected together and ready to be used we pressed the button "Record Trajectory" in GUI application and give a start command to a car, by controlling joystick, while the car is moving with a help of joystick position of the car is stored every few milliseconds. When "car's journey" is done and we want to store recorded and stored data to a file we again pressed the button "Save into the File" in GUI application and one trajectory is recorded. This process needed to be repeated every time when we want to record trajectory.
As mentioned above there are two map types: X and T intersections. For X-intersection a various number of trajectories for movement to the right, straight and left were recorded, while for T-intersection only movement t the right and left was recorded.

### 4.1.2 Data Collection from Real Car

TU-Darmstadt has autonomous driving working group aDDa, which is developing software and hardware to make a real autonomous car (Figure 4.3). We made test drives to collect data from a real car and ran our probabilistic intention prediction algorithm on this dataset.

**Figure 4.3:** On the left side is the picture shows how data collection looked like and the right side shows where ADMA sensor is in the car

We were interested in only ADMA data (ADMA - the high-precision gyro measuring system specially developed for vehicle dynamics measurements), which we recorded using ROS recording node while the car was moving. Figure 4.3 includes a picture of the test drive environment and ADMA position in the car. After recording data, we had a *.bag file with all information we can get from ADMA: acceleration, speed and position of moving vehicles in three-dimensional (x, y, z) axes. For our working purposes, we were only interested in position data, due to that the listener node, listening the only position was written (we run listener ROS node on *.bag information file from ADMA and we save all recorded positions to the *.csv file).

Real Car data, recorded with ADMA did not have the same start position (and not even close ones). Since this gave some inaccuracy for our algorithm, we converted data to a Cartesian coordinate system, with the starting point (14.5, 2.0) and then when a new position arrives, we computed the difference of the new position to the first real position (e.g. first three points for one trajectory after this changes look like this: [[14.5, 2.0], [14.7, 2.4], [14.9, 2.8], ....]
After these changes, we have a set of reasonably represented data from the real car.

## 4.2 Brief Algorithm Explanation

Following Chapter 3, the whole framework starts to work with map recognition, when the type of the map is known, initial belief ($b_0$) is calculated. When recognized map is X intersection initial belief is ($b_0$[right, straight, left] = $(0.333, 0.333, 0.333)$), when we are working with T intersection, initial belief is $b_0$[left, right] = $(0.533, 0.467)$. Map recognition is introduced in Chapter 3, Section 3.1. Once we know the map type we are preparing our demonstrations for learning initial probabilistic prediction model. Unification of all demonstrations in our DB is made using interpolation, detailed explanation is in Chapter 3, Section 3.2. After interpolation is done our algorithm is learning initial probabilistic model: in X intersection we are working with 3, while in T intersection with 2 movement classes. How algorithm is learning initial probabilistic prediction model is described in Chapter 3, Section 3.4. Section 3.4 in Chapter 3, explains how belief is calculated and updated. Prediction is making until trajectory is over, due to that after every step algorithm is checking if movement still happening, if yes, the last belief is updated and new prediction is made, while trajectory ends. If current time step is the last for trajectory then algorithm stops. The brief scheme of algorithm workflow is in Figure 4.4

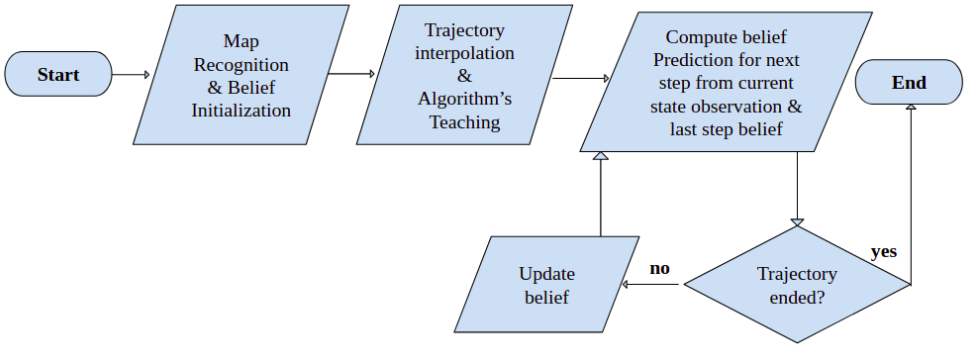


**Figure 4.4:** The workflow of the probabilistic intention prediction algorithm. Starting with the map recognition process, according to the map initial belief is initialized. After that trajectories, unification and initial prediction model creation happened. The last step when we have all this initial information is to make predictions for that we need last belief, current car position and learned data, when prediction is made, it is checking if trajectory is over, if yes, then algorithm is ending, if not, then cycle with updated belief and new observation is continuing

As an additional part of thesis trajectory scaling process was introduced and checked. Scaling was introduced following the common logic that making more belief updates, i.e. having more time steps in trajectory, would allow to have better precision. The problem with more time steps is the longer computation time, which in critical infrastructures, as vehicles, should be as small as possible. We wanted to achieve precision of 100 time steps, but having only 10 steps in trajectory.
On more additional part of thesis, when we managed to predict step by step movement, we wanted to see if we can predict all future trajectory. We tested that, using ProMPs and approach, proposed in [17].
The results for this and other methods, described in this chapter can be found in flowing Chapter 5 "**Experiments and Results**".

# 5 Experiments and Results

This chapter shows the results of the testings on proposed probabilistic intention prediction algorithm. Different trajectory sets were tested: the first one with **pre-recorded** and the second one with **real time** trajectories. Different setups implemented and evaluated with respect to classification of different movements and future trajectories prediction. This chapter is based on a number of experiments, where different trajectories for all movement classes were tested. Results were analyzed separately and then compared.

## 5.1 Future Trajectory Prediction, Using Pre-Recorded Data

This subsection presents results of the evaluation for the probabilistic intention prediction algorithm, which was described in Chapter 4. Experiments were made with prerecorded data and in two different environments: X and T intersections. The data was collected and recorded using ROS simulation, as introduced in Chapter 4, Section 4.1.

### 5.1.1 Experiments Made Using X-Intersection

The first step step of the proposed algorithm is to recognize the map (process of recognition is described in Chapter 3, Section 3.1. After knowing in which map environment we are working, it is important to obtain initial values, which are necessary for further calculations based on which prediction on movement is done.
From recorded DB, our algorithm learned a probabilistic prediction model (a mean trajectory and standard deviation on that trajectory). Figure 5.1 shows, how X intersection map looks like and illustrates a probabilistic prediction model, learned at the very beginning of the algorithm. The initial position of the car is $x_0, y_0 = (14.5, 2.0)$ and from there, having X intersection type map car can go to any direction it wants (i.e. right, straight or left). Another very important variable to know at the very beginning is the initial belief, which shows how likely car will go to any of possible directions. This value directly depends on the map type, and since now our map is X intersection which allows movement to all directions, our initial belief for right, straight and left classes is $b_0 = (0.333, 0.333, 0.333)$.

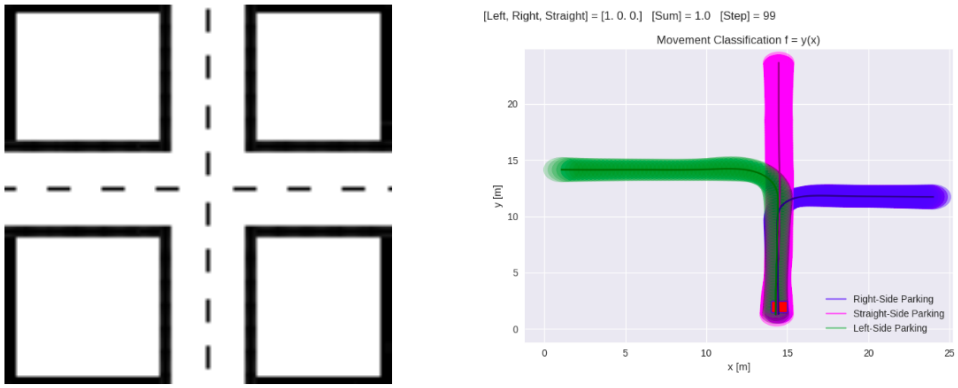


**Figure 5.1:** The left side of the figure shows how X-Intersection map, used for experiments, looks like. The right side of the figure shows the probabilistic prediction model, learned by algorithm: blue, magenta and green areas show the mean trajectory and standard deviation for respectively right, straight and left movement classes. The right side of the figure contains other important information: movement start position is $x_0, y_0 = (14.5, 2.0)$, illustrated with red rectangular and initial belief for each movement class with this setup is $b_0 = (0.333, 0.333, 0.333)$

#### Experiments Made Using X-Intersection. Movement Classification

In the X intersection section, we have 3 movement classes: right, straight and left, Figure 5.2 illustrates trajectories with which experiments was made.

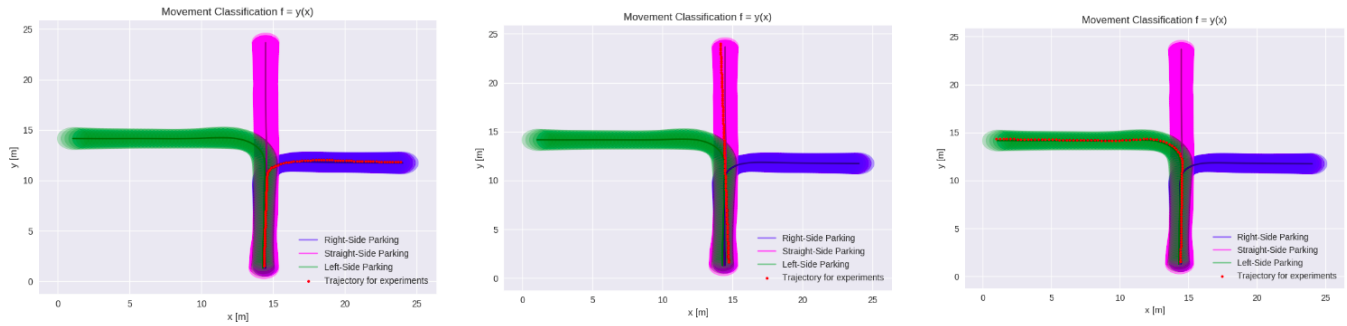


**Figure 5.2:** The figure shows, trajectories with which experiments were made. Testing trajectories are in red. The picture on the left shows a trajectory which classified as right, middle picture - as straight and the right picture as left movement class. Testing trajectories do not belong to the training data set

As mentioned above in order to test algorithm we are using interpolated trajectories since all of them are not equal length/time-step wise. Due to method introduced in Chapter 3, Subsection 3.2, the number of time steps can be chosen freely. In order to see if the number of time steps have any affect on results, we checked how much of trajectory need to be passes to the algorithm to get the correct prediction. Results of this experiment with movement to the right is in the table 5.1:

| Time Steps in Trajectories | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 200 |
| How many steps needed to go | 4 | 7 | 11 | 15 | 20 | 24 | 28 | 32 | 38 | 40 | 80 |
| How much trajectory was needed to see, % | 40 | 35 | 36.7 | 37.5 | 40 | 40 | 40 | 40 | 42.2 | 40 | 40 |

**Table 5.1:** Average time-steps needed to predict trajectory. The same trajectories were tested, just were interpolated differently. The second row, says after how many steps direction was recognized correctly (belief for going to that direction is >0.5 and after that step prediction for other directions do not occur). The third row shows how much trajectory needed to be seen to make a correct prediction. Trajectory direction is right

The table 5.2 summaries how prediction precision is affected by the number of time steps in trajectory while moving straight:

| Time Steps in Trajectories | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 200 |
| How many steps needed to go | 3 | 6 | 8 | 11 | 15 | 18 | 20 | 23 | 28 | 30 | 60 |
| How much trajectory was needed to see, % | 30 | 30 | 26.7 | 27.5 | 30 | 30 | 28.6 | 28.75 | 31.1 | 30 | 30 |

**Table 5.2:** Average time-steps needed to predict trajectory. The same trajectories were tested, just were interpolated differently. The second row, says after how many steps direction was recognized correctly (belief for going to that direction is >0.5 and after that step prediction for other directions do not occur). The third row shows how much trajectory needed to be seen to make a correct prediction. Trajectory direction is straight

The table 5.3 summaries how prediction precision is affected by the number of time steps in trajectory, while moving to the left:

| Time Steps in Trajectories | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 200 |
| How many steps needed to go | 3 | 7 | 10 | 14 | 17 | 20 | 23 | 28 | 30 | 31 | 60 |
| How much trajectory was needed to see, % | 30 | 35 | 33.3 | 35 | 34 | 33.3 | 32.9 | 35 | 33.3 | 31 | 30 |

**Table 5.3:** Average time-steps needed to predict trajectory. The same trajectories were tested, just were interpolated differently. The second row, says after how many steps direction was recognized correctly (belief for going to that direction is >0.5 and after that step prediction for other directions do not occur). The third row shows how much trajectory needed to be seen to make a correct prediction. Trajectory direction is left

From the results of tables 5.1, 5.2 and 5.3 we can see that our normalization method is working and number of times steps does not effect results so much. For the following experiments we choose to with 10 and 100 time steps.

**Movement Classification: Right**

To illustrate how probabilistic intention prediction algorithm is working, we used pre-recorded trajectory, interpolated it to 10 steps and made figure which illustrates prediction making results step-by-step.

Figure 5.3 shows these results. From the series of plots, we can see that in the 5th step algorithm beliefs that direction is right (result is equals to 1). In the figure step number is 4, because counting starts with 0. Before 5th steps algorithm was also thinking that a car can go straight, it happened because values of standard deviation for testing trajectory was very closer to straight class than right or left. After making the 5th step algorithm knew for sure that movement direction is right since values for straight and left directions were very far away from the current car position.

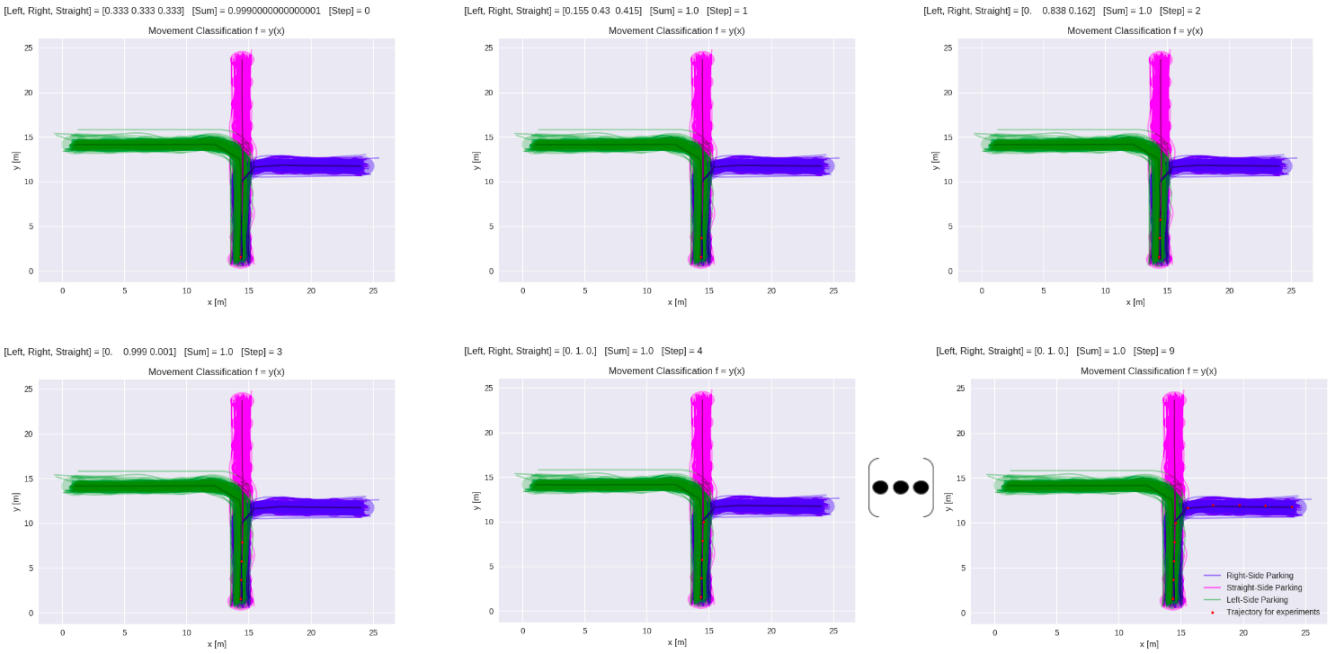


**Figure 5.3:** Process of prediction making for trajectory which belongs to the right movement class. Trajectory has 10-time steps (counting from 0 to 9) and results after each time step are shown separately. After Step = 4 prediction that movement is to the right is equal to 1.0, due to that lack of space and the same result, we skipped to show step = 5 - 8

While making experiments, there were a lot of trajectory tested. We noticed that even if trajectories belong to the same movement class, all of them are different (e.g. starting point slightly changes, some turnings are not so smooth as others, etc). All this results in being recognized correctly in a different time: "better" trajectories (e.g. the ones which are closer to the mean trajectory of movement class) are predicted correctly faster than the ones which are "worse" (e.g. making movement for one or another direction, not keeping straight line, etc.). To be able to see how belief changing over time for different trajectories and visually to see which trajectory is recognize faster than another Figure 5.4 was made. This figure was created just to show how fast (time step wise) trajectories are predicted correctly, figure does not show how

each trajectory looks like. Figure 5.4 contains 10 random different trajectories of right movement class (which does not belong to training data set), interpolated for 100-time steps each.

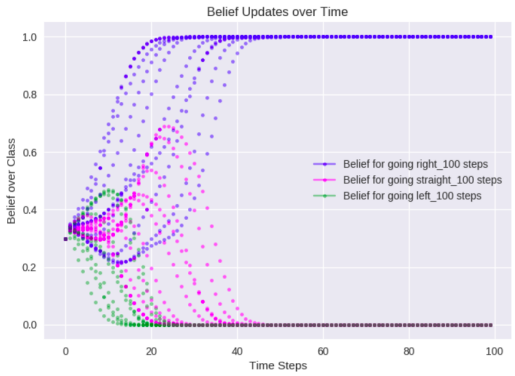


**Figure 5.4:** Belief changes over time. 10 random trajectories (which does not belong to training data set) were interpolated for 100-time steps and tested with probabilistic intention prediction algorithm to see what is the pattern of for recognizing trajectories. Results shows that all trajectories were recognized correctly, just for some of them it took more time to be predict correctly. All tested trajectories belong to the same movement class. Movement class is right

Figure 5.4 illustrates that some trajectories required more time to be recognized correctly. As mentioned before, from this figure it is not possible to see and understand how *the fastest* and *the slowest* to recognize trajectories look like. For this purpose two more figures were computed: Figure 5.5 shows how *the fastest* trajectory to recognize look like and how its belief changes over time and Figure 5.6 shows how *the slowest* trajectory to recognize looks like.

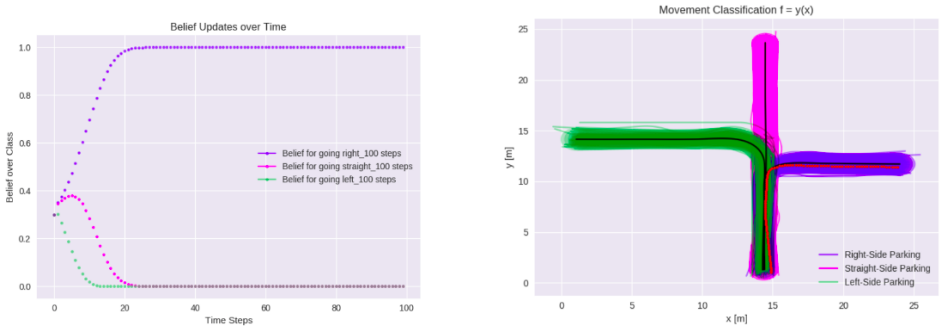


**Figure 5.5:** The fastest trajectory to recognize, taken from Figure 5.4. On the left side of the figure belief changing over time is shown, from it is possible to see that trajectory is correctly recognized within 20 steps (out of 100). The right image shows how the fastest trajectory to recognize looks like. Trajectory direction is right

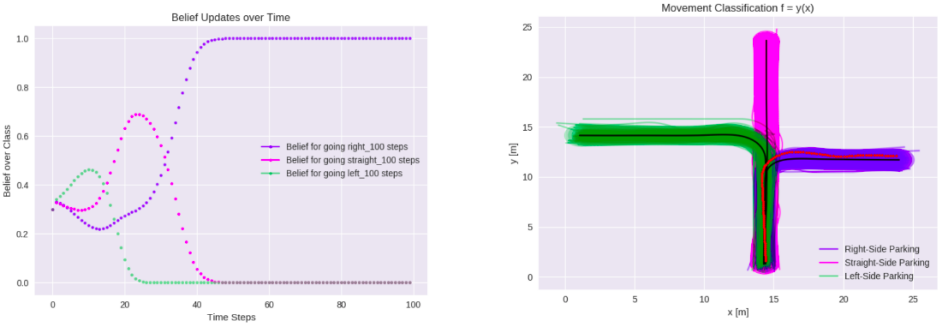


**Figure 5.6:** The slowest trajectory to recognize, taken from Figure 5.4. On the left side of the figure belief changing over time is shown, from it is possible to see that trajectory is correctly recognized within more than 40 steps (out of 100). The right image shows how the slowest trajectory to recognize looks like. Trajectory direction is right

As mentioned before *the slowliness* in recognition can be explained by the position of the trajectory: it is close to all means, so it is natural that precise of prediction making can be disturbed in this case.
By looking at results from *the fastest* trajectory to recognize, we can also make an assumption that trajectory is not on the all means, but it is closer to the standard deviation value of the right class and due to that precise of prediction making is better in this case.

**Movement Classification: Straight**

Middle picture of Figure 5.2 shows how straight movement class looks like. The table 5.2 shows the prediction preciseness depending on time steps in trajectory. As shown in a paragraph about the right movement class, Figure 5.7 shows prediction making process, just this time for a straight movement class. From plots we can see that straight test trajectory is also easy to recognize: after 5th step, a prediction that car is going straight is equal to 0.997 and only growing while time passing.

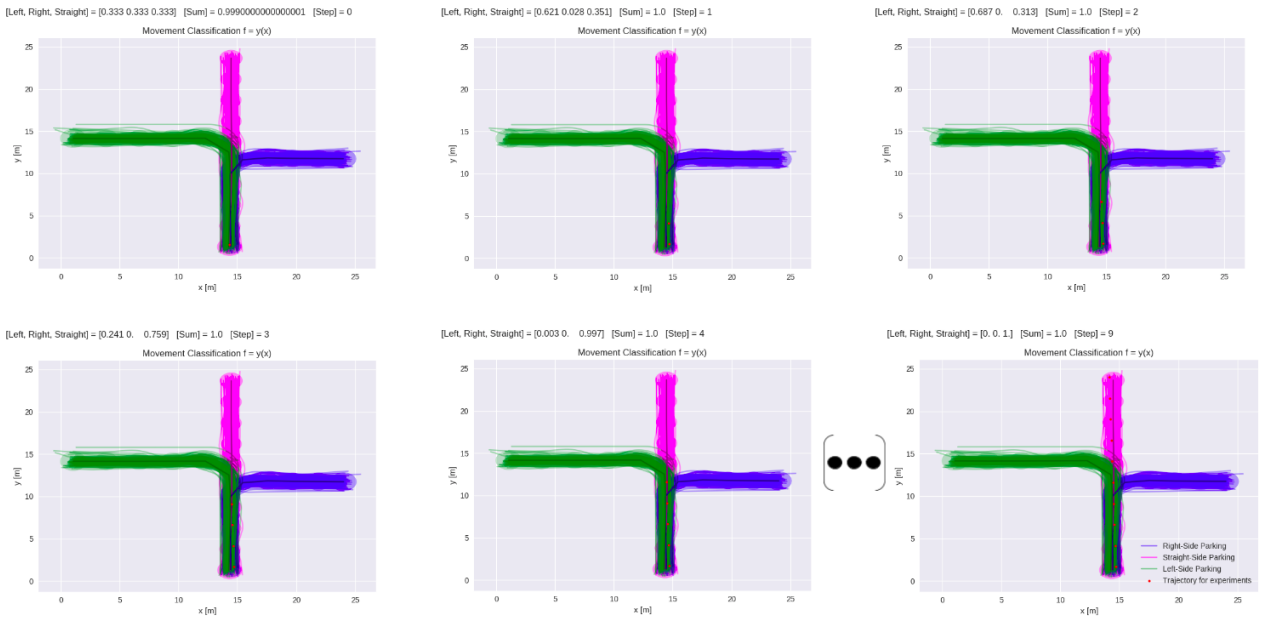


**Figure 5.7:** Process of prediction making for trajectory which belongs to the straight movement class. Trajectory has 10-time steps (counting from 0 to 9) and results after each time step are shown separately. After Step = 4 prediction that movement is to straight is equal to 0.997 and only getting bigger, due to that lack of space and the same result, we skipped to show step = 5 - 8

Straight movement class also has different trajectories, and different time is needed to predict movement correctly. Figure 5.8 shows 10 random trajectories (which do not belong to algorithm training data set) and their belief updates over time. Figure 5.9 and Figure 5.10 respectively shows *the fastest* and *the slowest* trajectory to recognize. Trajectories were taken from the same set as in Figure 5.8.

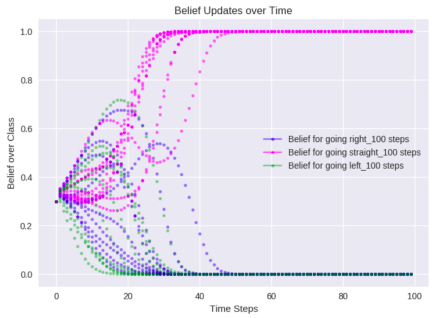


**Figure 5.8:** Belief changes over time. 10 random trajectories (which does not belong to training data set) were interpolated for 100-time steps and tested with movement recognition algorithm to see which trajectory is predicted correctly faster and which slower. All tested trajectories belong to the same movement class. Movement class is straight

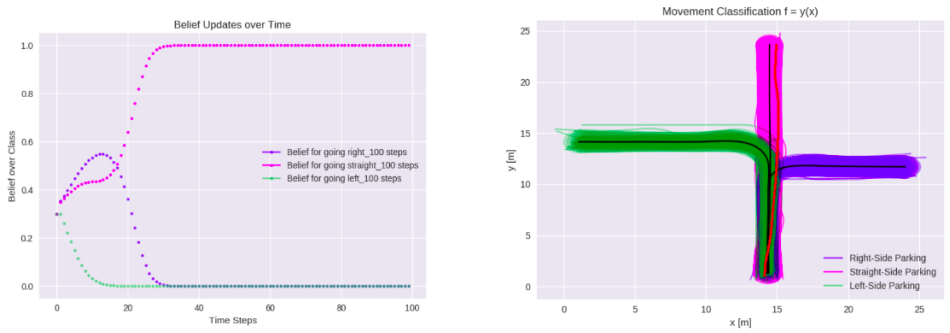


**Figure 5.9:** The fastest trajectory to predict correctly, taken from Figure 5.8. On the left side of the figure belief changing over time is shown, from it is possible to see that trajectory is correctly recognized within 30 steps (out of 100). The right image shows how the best trajectory looks like. Trajectory direction is straight

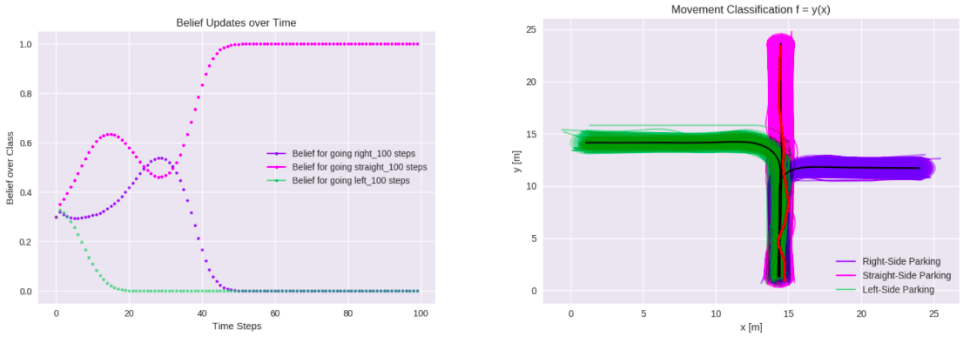


**Figure 5.10:** The slowest trajectory to be predicted correctly, taken from Figure 5.8. On the left side of the figure belief changing over time is shown, from it is possible to see that trajectory is correctly recognized within more than 50 steps (out of 100). The right image shows how the worse trajectory looks like. Trajectory direction is straight

By looking at Figure 5.10, *the long* time for being recognized and predicted correctly can be explained by the position of the trajectory: it is closer to standard deviation values of left class mean, this could be the case, why prediction is that car is going to the left, until it is sure that it is not going to the left for sure.
By looking at Figure 5.9 *the fastest* recognition can also be explain with assumption that trajectory and coordinates at each time step are closer to mean and it is in the range of standard deviation of the straight mean.

**Movement Classification: Left**
Example of how left movement class looks like it is possible to find in the same Figure 5.2 together with straight and right movement classes. The table 5.3 shows the prediction preciseness depending on time steps in trajectory. Figure 5.11

represents set of plots which represent prediction making process step by step. As usual test trajectory is interpolated to have 10 time steps from results we can see that trajectory which belongs to the left movement class is very easily recognizable: after 3rd step, a prediction that car is going straight is equal to 0.999 and only growing while time passing.

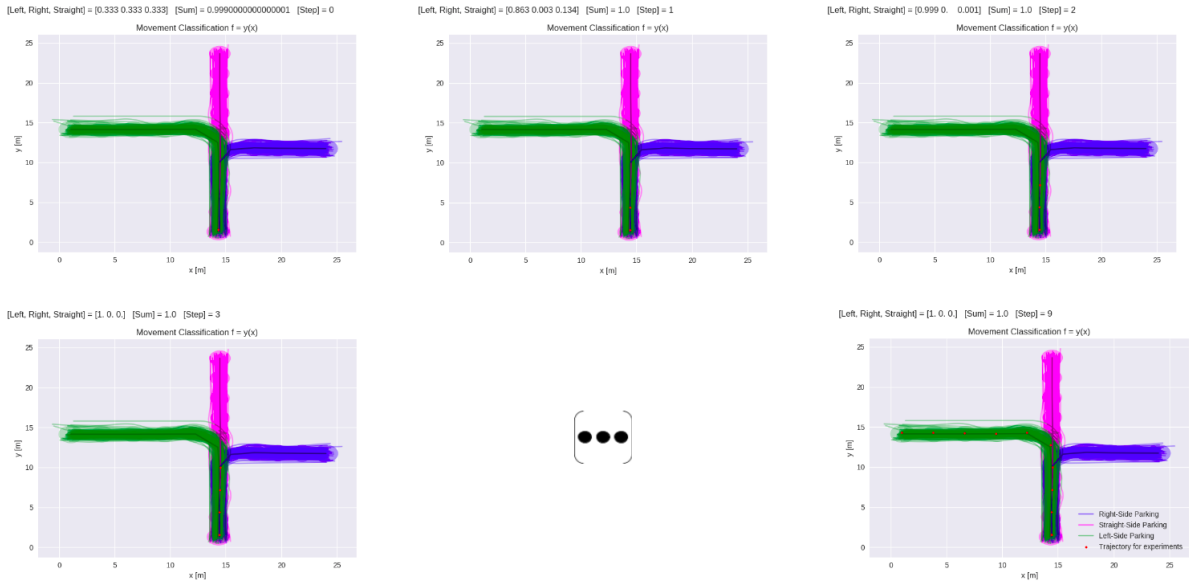


**Figure 5.11:** Process of prediction making for trajectory which belongs to the straight movement class. Trajectory has 10-time steps (counting from 0 to 9) and results after each time step are shown separately. After Step = 2 prediction that movement is to straight is equal to 0.999 and only getting bigger, due to that lack of space and the same result, we skipped to show step = 4 - 8

Left movement class, as well as other movement classes, have trajectories which are predicted correctly not within the same amount of time steps. Figure 5.12 shows 10 random trajectories (which do not belong to algorithm training data set) and their belief updates over time. Figure 5.13 and Figure 5.14 respectively shows *the fastest* and *the slowest* trajectories to recognize, taken from the set of trajectories in Figure 5.12.

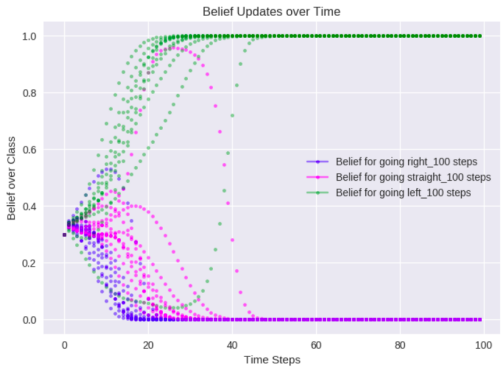


**Figure 5.12:** Belief changes over time. 10 random trajectories (which does not belong to training data set) were interpolated for 100-time steps and tested with movement recognition algorithm to see which trajectory is easier recognized (better) and which one is more difficult (worse) to recognize correctly. All tested trajectories belong to the same movement class. Movement class is left

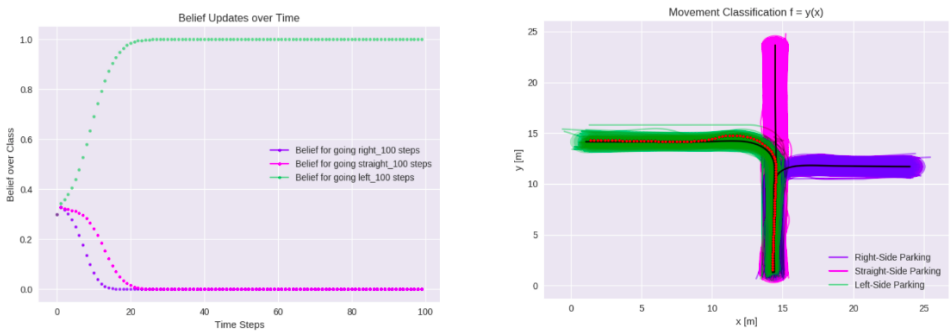


**Figure 5.13:** The fastest recognizable trajectory, taken from Figure 5.12. On the left side of the figure belief changing over time is shown, from it is possible to see that trajectory is correctly recognized within the first 20 steps (out of 100). The right image shows how the best trajectory looks like. Trajectory direction is left

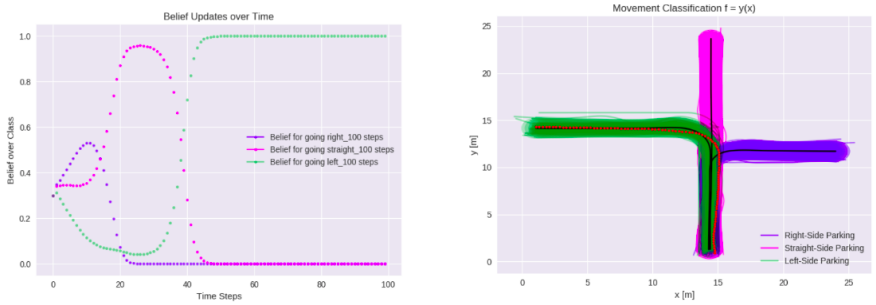


**Figure 5.14:** The slowest recognizable trajectory, taken from Figure 5.12. On the left side of the figure belief changing over time is shown, fromit is possible to see that trajectory is correctly recognized within more than 50 steps (out of 100). The right image shows how the worse trajectory looks like. Trajectory direction is left

### 5.1.2 Experiments Made Using T-Intersection

As mentioned before we compute a probabilistic prediction model from demonstrations, depending on the map type probabilistic prediction model differs. T intersection has one direction less than X intersection, changes values of our model. Together with values which differs is initial belief $b_0$, for right movements class $b_0 = 0.467$ and for left is equal to $b_0 = 0.533$ (calculated using formulas from Chapter 3). The initial starting position is the same: $x_0, y_0 = (14.5, 2.0)$. Figure 5.15 shows initial how tested map looks like and information which we have at the very beginning of the algorithm.

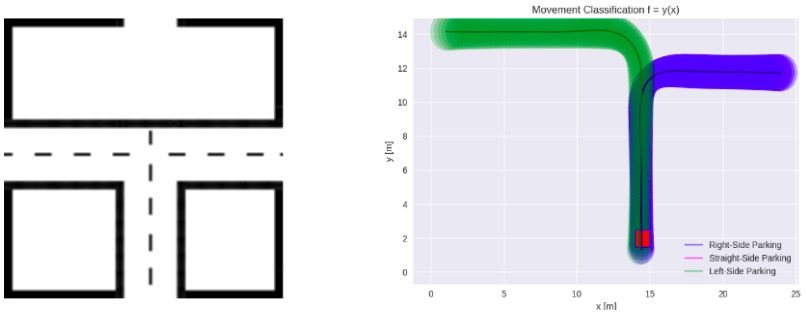


**Figure 5.15:** The left side of the figure shows how T-Intersection map, used for experiments, looks like. The right side of the figure shows the probabilistic prediction model, learned by algorithm: blue and green areas show the mean trajectory and standard deviation for respectively right and left movement classes. The right side of the figure contains other important information: movement start position is $x_0, y_0 = (14.5, 2.0)$ and initial belief for each movement class, which is $b_0$[left, right] $= (0.533, 0.467)$

T intersection has 2 movement classes: right and left, Figure 5.16 shows random testing trajectories (which do not belong to training data set) for each movement class.

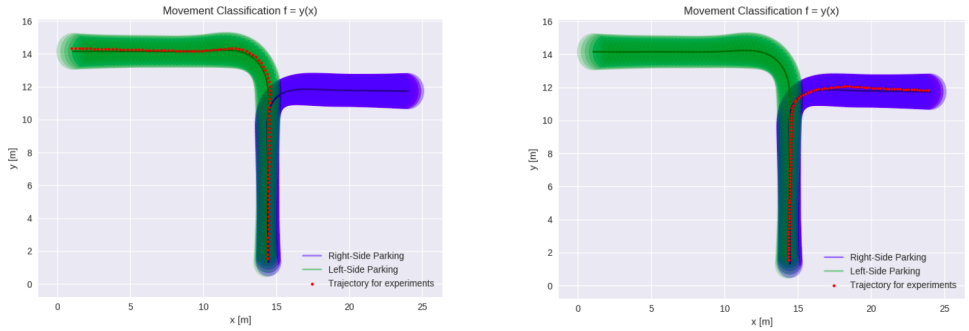


**Figure 5.16:** The figure shows random testing trajectories in red for each movement class. The picture on the left shows a trajectory from the right class and the right picture represents left movement class. Testing trajectories do not belong to the training data set

Precision of Prediction and dependence from time steps was tested with T intersection as well, results are in tables 5.4 and 5.5.

| Time Steps in Trajectories | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 200 |
| How many steps needed to go | 2 | 4 | 7 | 10 | 12 | 15 | 16 | 19 | 22 | 25 | 40 |
| How much trajectory was needed to see, % | 20 | 20 | 23.3 | 25 | 24 | 25 | 22.9 | 23.75 | 24.4 | 25 | 25 |

**Table 5.4:** Average time-steps needed to predict trajectory. The same trajectories were tested, just were interpolated differently. The second row, says after how many steps direction was recognized correctly (belief for going to that direction is >0.5 and after that step prediction for other directions do not occur). The third row shows how much trajectory needed to be seen to make a correct prediction. Trajectory direction is right

| Time Steps in Trajectories | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 200 |
| How many steps needed to go | 2 | 4 | 6 | 9 | 11 | 13 | 15 | 19 | 19 | 25 | 50 |
| How much trajectory was needed to see, % | 20 | 20 | 20 | 22.5 | 22 | 21.7 | 30 | 23.75 | 21.1 | 25 | 25 |

**Table 5.5:** Average time-steps needed to predict trajectory. The same trajectories were tested, just were interpolated differently. The second row, says after how many steps direction was recognized correctly (belief for going to that direction is >0.5 and after that step prediction for other directions do not occur). The third row shows how much trajectory needed to be seen to make a correct prediction. Trajectory direction is left

As in X intersection case we can see that our normalization method is working and number of times steps does not effect results so much. For the following experiments we choose to with 10 and 100 time steps. Following the same logic for result presentation as in X intersection is used.

**Movement Classification: Right**

As just mentioned, an example of how the right movement class look like in T intersection environment is shown on the left side of Figure 5.16.

This paragraph will provide the same figures just in different environmental set up as while working with X-intersection.

Table 5.12 shows prediction results of each step:

| Belief that Movement class is ... | | | |
|---|---|---|---|
| Time Step | Left | Right | Straight |
| 0 | 0.524 | 0.476 | 0.0 |
| 1 | 0.284 | 0.716 | 0.0 |
| 2 | 0.010 | 0.980 | 0.0 |
| 3 - 9 | 0.0 | 1.0 | 0.0 |

**Table 5.6:** Belief Update Over Time for straight trajectory. Algorithm knew to which direction car is going from the very beginning

To be able to see how belief changing over times look for different trajectories and to differentiate which trajectories are *faster* recognizable than another Figure 5.17 was created. Here 10 random trajectories (which do not belong to the training dataset) are plotted.

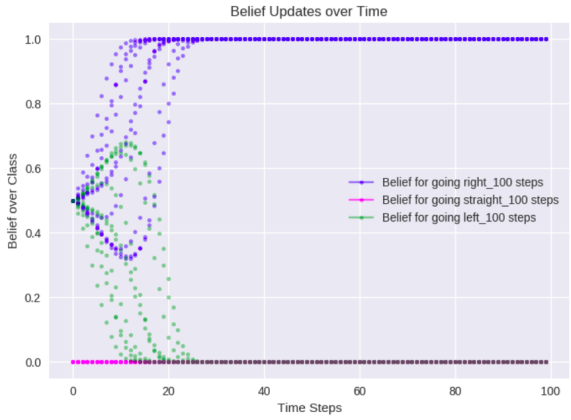


**Figure 5.17:** Belief changes over time. 10 random trajectories (which does not belong to training data set) were interpolated for 100-time steps and tested with movement recognition algorithm to see which trajectory is faster and which is slower recognizable. All tested trajectories belong to the same movement class. Movement class is right

To see how *the fastest* and *the slowest* trajectories to recognize look like Figure 5.18 and Figure 5.19 respectively were created.

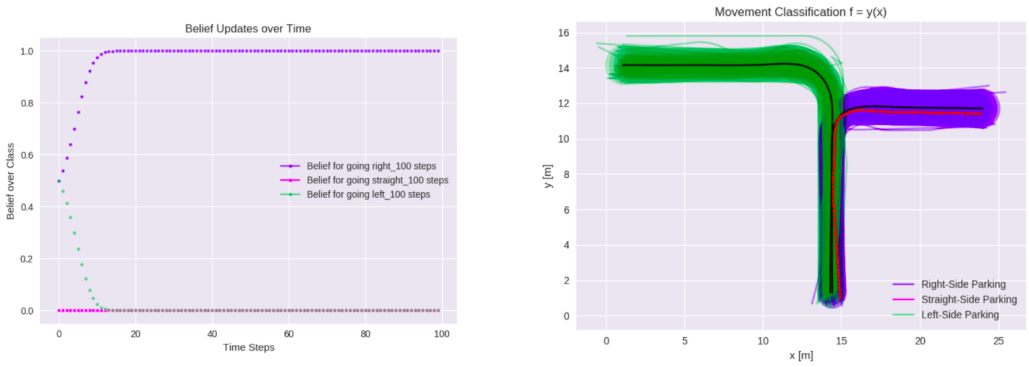


**Figure 5.18:** The fastest trajectory to recognize, taken from Figure 5.12. On the left side of the figure belief changing over time is shown, from it it is possible to see that trajectory is correctly recognized within the first 10 steps (out of 100). The right image shows how the best trajectory looks like. Trajectory direction is right

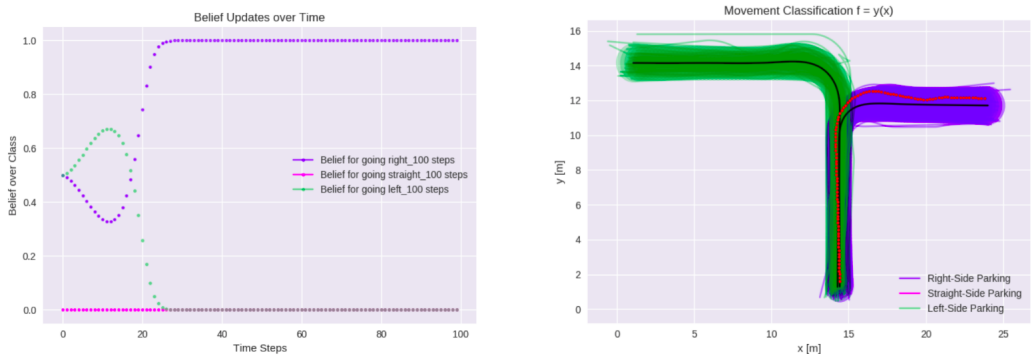


**Figure 5.19:** The slowest trajectory in correct recognition, taken from Figure 5.12. On the left side of the figure belief changing over time is shown, from it it is possible to see that trajectory is correctly recognized within more than 30 steps (out of 100). The right image shows how the worse trajectory looks like. Trajectory direction is right

From the Figure 5.17 it is possible to see that correct prediction is made much faster than in case of an X-Intersection map (Figure 5.4). This happened because of prior information and initial belief calculated according to it.

**Movement Classification: Left**

An example of how the left movement class look like in T intersection environment is shown together with example of left movement class, just on the right side of Figure 5.16.

Table 5.7 shows prediction results of each step:

| Belief that Movement class is ... | | | |
|---|---|---|---|
| Time Step | Left | Right | Straight |
| 0 | 0.524 | 0.476 | 0.0 |
| 1 | 0.997 | 0.003 | 0.0 |
| 2 - 9 | 1.0 | 0.0 | 0.0 |

**Table 5.7:** Belief Update Over Time for straight trajectory. Algorithm knew to which direction car is going from the very beginning

To be able to see how beliefs are changing over time withing different trajectories and compare how the movement recognition process looks we plotted 10 random trajectories going to the right. Results are shown in Figure 5.20.

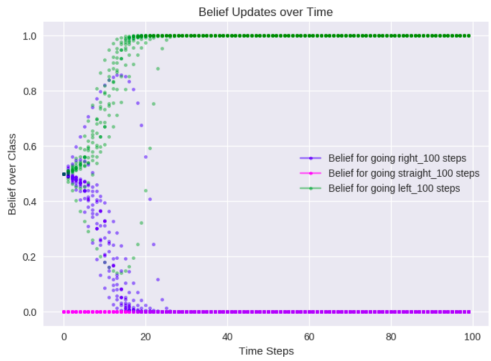


**Figure 5.20:** Belief changes over time. 10 random trajectories (which does not belong to training data set) were interpolated for 100-time steps and tested with movement recognition algorithm to see which trajectory is faster and which is slower recognizable. All tested trajectories belong to the same movement class. Movement class is left

Figure 5.21 and Figure 5.22 show how *the fastest* and *the slowest*, respectively, trajectories to recognize look like

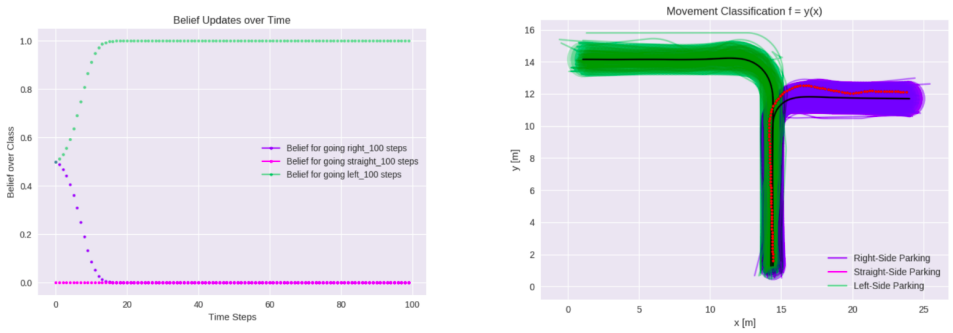


**Figure 5.21:** The fastest trajectory to recognize, taken from Figure 5.12. On the left side of the figure belief changing over time is shown, from it it is possible to see that trajectory is correctly recognized within the first 20 steps (out of 100). The right image shows how the best trajectory looks like. Trajectory direction is left

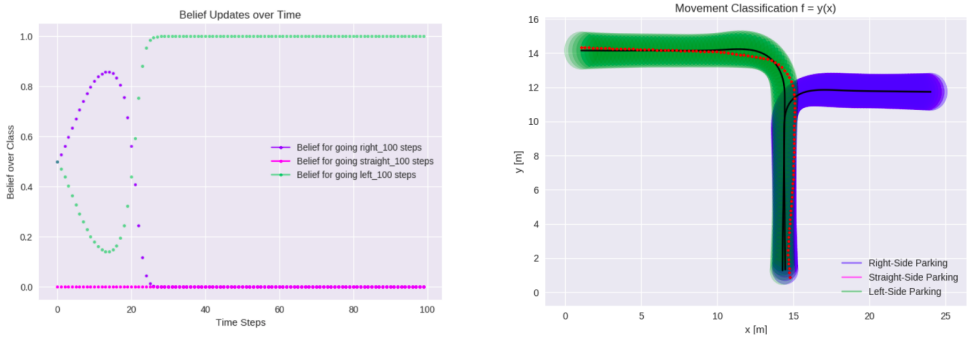


**Figure 5.22:** The slowest trajectory to recognize, taken from Figure 5.12. On the left side of the figure belief changing over time is shown, from it it is possible to see that trajectory is correctly recognized within more than 30 steps (out of 100). The right image shows how the worse trajectory looks like. Trajectory direction is left

### Experiments Using Pre-Recorded Data. Comparison of Results in Different Environments

Summarizing all results, described before, we can state that all trajectories are recognized and predicted correctly. Due to the shape and "smoothness" of trajectory one trajectories are recognized faster than another but after comparing the average number of points for differently interpolated trajectories we can say, that independently from the number of points in the trajectory and from direction class around 40% of trajectory needed to be seen to recognize trajectory correctly. A summary of X and T intersections are respectively in the table 5.8 and 5.9

| Movement Class | | | |
|---|---|---|---|
| | Right | Straight | Left |
| How much trajectory was needed to see | about 40% | about 20% | about 33% |

**Table 5.8:** Average length of trajectory needed to predict movement class correctly in X intersection

| Movement Class | | |
|---|---|---|
| | Right | Left |
| How much trajectory was needed to see | about 20% | about 25% |

**Table 5.9:** Average length of trajectory needed to predict movement class correctly in T intersection

Prediction is faster in T intersection because of the initial belief $b_0$, which belong to probabilistic prediction model learned from demonstrations, while in X intersection assumption about initial belief was that it is an equal possibility that car will go to any possible direction.

Below in the tables 5.10 and 5.11 of the maximum, average and minimum length of trajectory, which was necessary for correct movement direction prediction.

| Movement Class | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Right | | | Straight | | | Left | | |
| Max Len. | Avg. Len. | Min Len. | Max Len. | Avg. Len. | Min Len. | Max Len. | Avg. Len. | Min Len. |
| 50% | about 40% | 25% | 40% | about 30% | 10% | 42% | about 33% | 15% |

**Table 5.10:** Maximum, average and minimum length of trajectory needed to predict movement class correctly in X intersection per each movement class

| Movement Class | | | | | |
|---|---|---|---|---|---|
| Right | | | Left | | |
| Max Len. | Avg. Len. | Min Len. | Max Len. | Avg. Len. | Min Len. |
| 25% | about 15% | 25% | 30% | about 25% | 8% |

**Table 5.11:** Maximum, average and minimum length of trajectory needed to predict movement class correctly in T intersection per each movement class

### 5.1.3 Trajectory Scaling

As we noticed from results before the pattern of recognizing movement for the trajectory is similar when trajectory has 10 and 100 time steps. But for obvious reasons making a prediction for a trajectory with 100 time steps is more precise, since it is doing more updates, but at the same time to make a prediction for trajectory with 100 time steps is more difficult computation-wise, since calculations need to be done 10 times more (and faster) in order to keep a prediction pace while car is moving. To solve this problem we asked ourselfs is it possible to get a prediction precision of 100 time steps while making calculations only 10 times?
To make this possible into our prediction making algorithm we included scaling function, which is described before and in Figure 3.8 it is possible to see the pseudo code for this process. To show the difference between scaled results and not scaled results two different graphs will be shown for one test.

#### Trajectory Scaling. X-Intersection. Movement Classes

X-Intersection has 3 different movement directions and figures below show the trajectory prediction for all 3 classes before and after scaling.

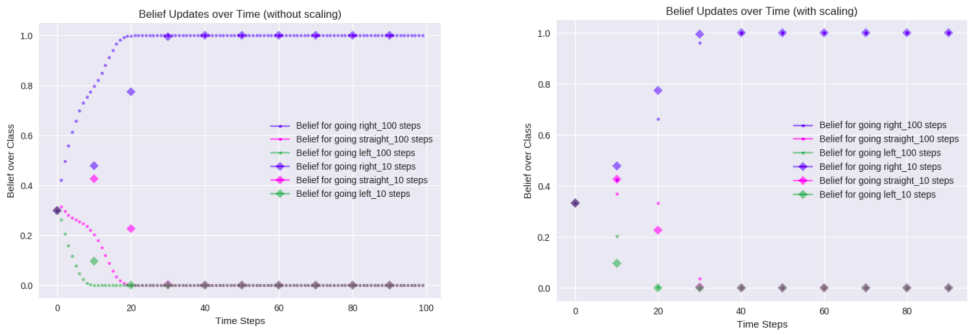


**Figure 5.23:** Belief updates over time. The picture on the left side of the figure shows the original belief update for the same trajectory interpolated for 10 and 100 time steps. The right side of the figure shows the prediction making process including scaling for the same trajectory interpolated for 10 and 100 time steps. For better visibility from trajectory with 100 time steps was printed only 10 points, matching points in from trajectory with 10 time steps. Trajectory direction is right

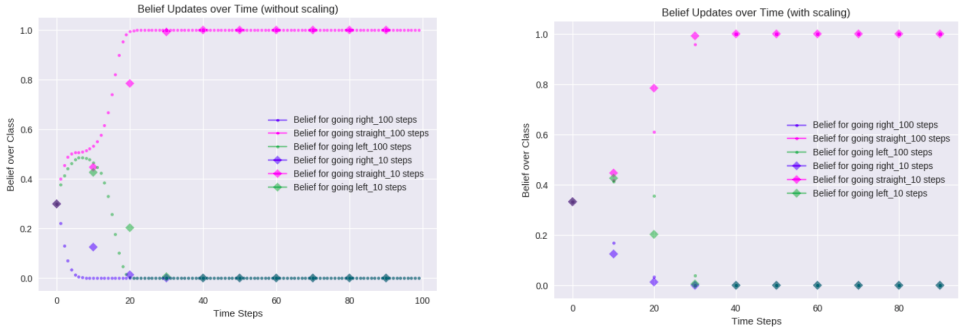


**Figure 5.24:** Belief updates over time. The picture on the left side of the figure shows the original belief update for the same trajectory interpolated for 10 and 100 time steps. The right side of the figure shows the prediction making process including scaling for the same trajectory interpolated for 10 and 100 time steps. For better visibility from trajectory with 100 time steps was printed only 10 points, matching points in from trajectory with 10 time steps. Trajectory direction is straight

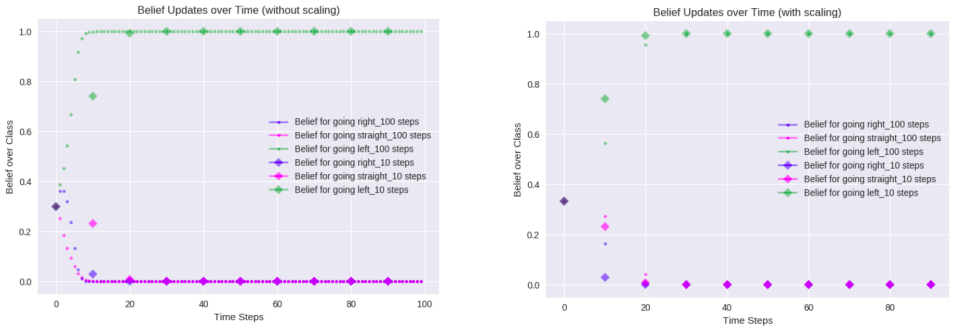


**Figure 5.25:** Belief updates over time. The picture on the left side of the figure shows the original belief update for the same trajectory interpolated for 10 and 100 time steps. The right side of the figure shows the prediction making process including scaling for the same trajectory interpolated for 10 and 100 time steps. For better visibility from trajectory with 100 time steps was printed only 10 points, matching points in from trajectory with 10 time steps. Trajectory direction is left

## Trajectory Scaling. T-Intersection. Movement Classes

T-Intersection has 2 different movement directions: right and left. Figures below show the trajectory prediction for all classes before and after scaling.

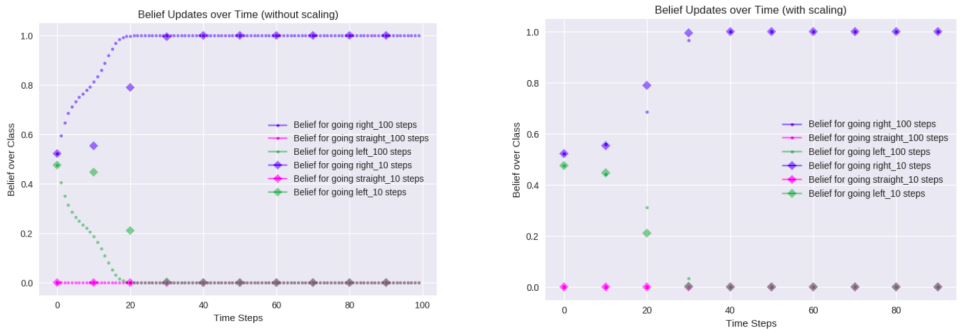


**Figure 5.26:** Belief updates over time. The picture on the left side of the figure shows the original belief update for the same trajectory interpolated for 10 and 100 time steps. The right side of the figure shows the prediction making process including scaling for the same trajectory interpolated for 10 and 100 time steps. For better visibility from trajectory with 100 time steps was printed only 10 points, matching points in from trajectory with 10 time steps. Trajectory direction is right

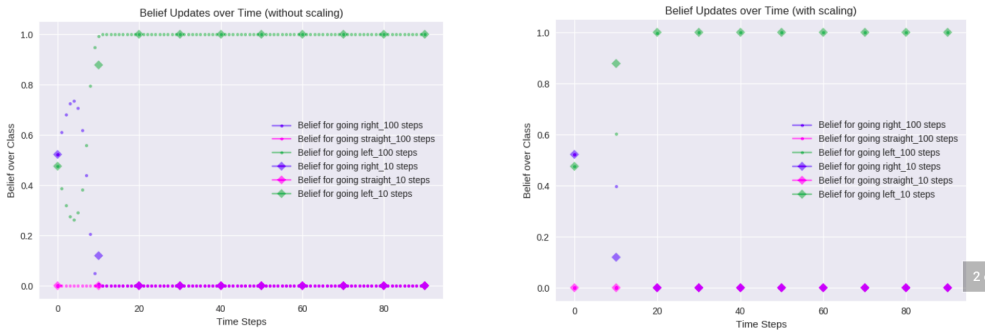


**Figure 5.27:** Belief updates over time. The picture on the left side of the figure shows the original belief update for the same trajectory interpolated for 10 and 100 time steps. The right side of the figure shows the prediction making process including scaling for the same trajectory interpolated for 10 and 100 time steps. For better visibility from trajectory with 100 time steps was printed only 10 points, matching points in from trajectory with 10 time steps. Trajectory direction is left

From the results for X and T intersections, we can see that the process of scaling is working and it gives much closer results between differently interpolated trajectories. Scaling can be used when for some reasons it is not possible to compute predictions as often. But for critical situations we still suggest to used not scaled method and use trajectory interpolated for more time steps.

### 5.1.4 Full Trajectory Prediction, using ProMPs

After being able to correctly predict future motions and intentions, we wanted to see if it is possible to predict how full trajectory looks like, having early observations of the car position and from demonstrations learned probabilistic prediction model, for this part, we decided to use Probabilistic Movement Primitive method ProMPs. As mentioned in Chapter 3, Section 3.6.1, where the theoretical background for ProMPs is covered, one of the most important variables for making a correct prediction is the number of RBFs. We for making finding the right number for it we used error between real and reconstructed trajectory dependence from a number of RBFs (more details given in the chapter, mentioned before). Looking into the error graph used 6 RBFs. The first thing which we did, having known weighting function and expression of RBFs, we reconstructed, how mean trajectory looks like. Figure 5.28 shows it.

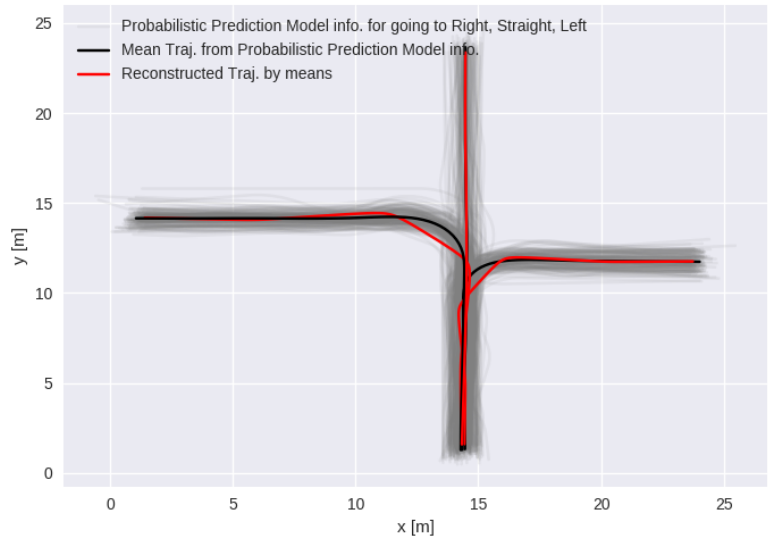


**Figure 5.28:** Reconstructed trajectory by means from probabilistic prediction model. All pre-recorded demonstrations and **mean trajectory**, from probabilistic prediction model

## 5.2 Experiments Using The Real Car Data

In previous chapters, data collection from a real car and its preparation for work was described. In this section results with real data is described.
Due to some technical problems, not all data was good for using, and at the end, we have 15 trajectories for algorithm learning (5 per each movement class) and 3 trajectories for testing (1 per each class). An algorithm training set is shown in Figure 5.29. We will choose trajectory randomly and will make all experiments with that trajectory. Only X intersection will be used for this testing.

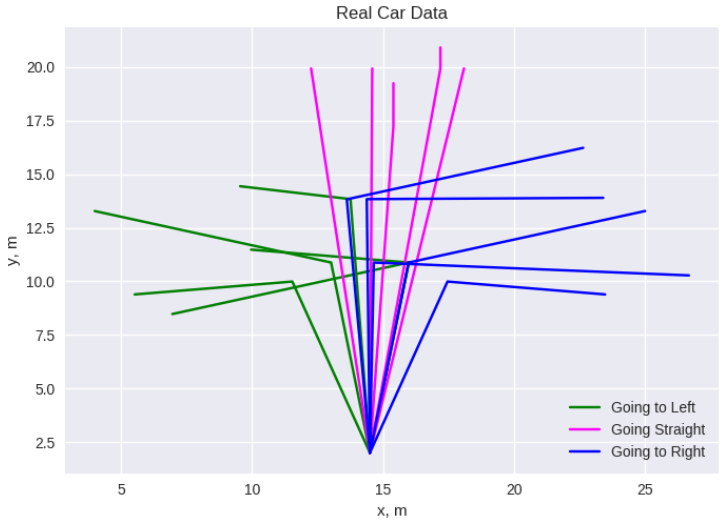


**Figure 5.29:** Trajectories recorded with a real aDDa car. The duration of plotted trajectories is 9 seconds each. Length of trajectories does not match for several reasons: because of duration unification, car speed and different real trajectory length (one turn was longer, another shorter, etc.)

For this real car data test, the same algorithm from before was used with different learning and testing data. The first run gave these results: Figure 5.30 and Table 5.12.

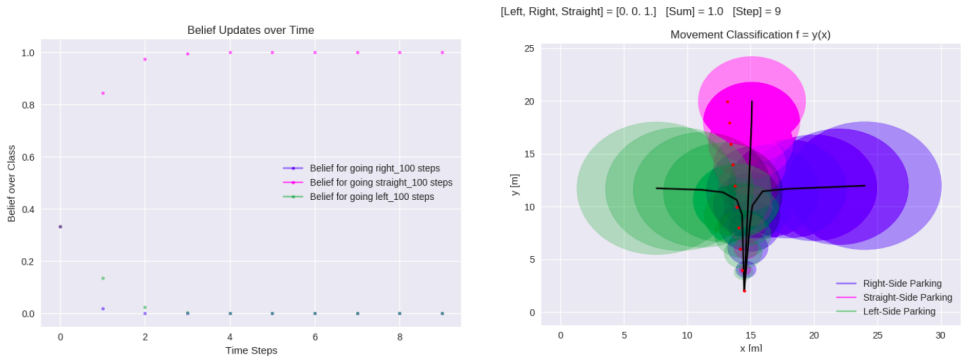


**Figure 5.30:** The image on the left side of the figure shows belief updates over time and on the right red dotted line is the tested trajectory, interpolated for 10 time steps

Table 5.12 shows prediction results of each step:

| Belief that Movement class is ... | | | |
|---|---|---|---|
| Time Step | Right | Left | Straight |
| 0 | 0.333 | 0.333 | 0.333 |
| 1 | 0.135 | 0.020 | 0.846 |
| 2 | 0.024 | 0.001 | 0.975 |
| 3 | 0.006 | 0.001 | 0.993 |
| 4 - 7 | 0.001 | 0.0 | 0.999 |
| 8 - 9 | 0.0 | 0.0 | 1.0 |

**Table 5.12:** Belief Update Over Time for straight trajectory. Algorithm knew to which direction car is going from the very beginning

Results of the second run is summarized in Figure 5.31 and Table 5.13.

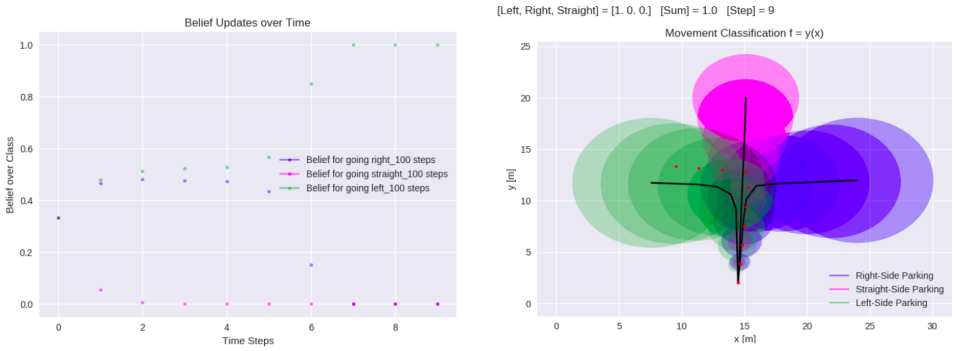


**Figure 5.31:** The image on the left side of the figure shows belief updates over time and on the right red dotted line is the tested trajectory, interpolated for 10 time steps

Table 5.13 shows prediction results of each step:

| Belief that Movement class is ... | | | |
|---|---|---|---|
| Time Step | Right | Left | Straight |
| 0 | 0.333 | 0.333 | 0.333 |
| 1 | 0.479 | 0.465 | 0.056 |
| 2 | 0.511 | 0.481 | 0.007 |
| 3 | 0.522 | 0.476 | 0.001 |
| 4 | 0.527 | 0.472 | 0.001 |
| 5 | 0.566 | 0.433 | 0.001 |
| 6 | 0.848 | 0.151 | 0.001 |
| 7 | 0.999 | 0.001 | 0.000 |
| 8 - 9 | 1.0 | 0.0 | 0.0 |

**Table 5.13:** Belief Update Over Time for left trajectory. Algorithm was pretty sure to which direction car is going from the very beginning

The last run looked like in Figure 5.32 and in the Table 5.14.

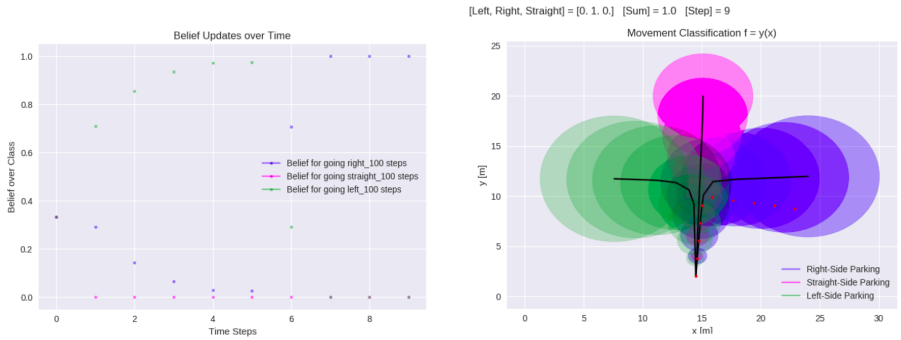


**Figure 5.32:** The image on the left side of the figure shows belief updates over time and on the right red dotted line is the tested trajectory, interpolated for 10 time steps

Table 5.14 shows prediction results of each step:

| Belief that Movement class is ... | | | |
|---|---|---|---|
| Time Step | Right | Left | Straight |
| 0 | 0.333 | 0.333 | 0.333 |
| 1 | 0.709 | 0.290 | 0.001 |
| 2 | 0.855 | 0.144 | 0.001 |
| 3 | 0.934 | 0.065 | 0.001 |
| 4 | 0.971 | 0.028 | 0.001 |
| 5 | 0.973 | 0.267 | 0.0 |
| 6 | 0.292 | 0.707 | 0.0 |
| 7 | 0.001 | 0.099 | 0.0 |
| 8 - 9 | 0.0 | 1.0 | 0.0 |

**Table 5.14:** Belief Update Over Time for right trajectory. While recognizing right trajectory algorithm thought that car is moving to left direction, but at the end it recognized direction correctly

From the results, we can see that the algorithm is working on not only on simulated data but with real information from the car. Results showed that some trajectories were recognized easier than another, unfortunately, due to the lack of testing trajectories, there was no possibility to test different trajectories and to get average results of real trajectories and to understand exactly why this happened. But from experiments with simulated data, we can make an assumption that tested data for right trajectory was not a really smooth one.

### 5.2.1 The Whole Trajectory Reconstruction using ProMPs

The explanation of ProMPs is done in the previous chapter. In this set of experiment, we will check how well ProMPs method can recreate a trajectory having a number of RBF and weights for each function, which was computed from the learning data set. As mentioned in the description chapter of ProMPs, it is very important to select the right number of RBF. As in the chapter of explanation of the method, the right number of RBF we selected by calculating the error of real and reconstructed trajectories (Figure 5.33).

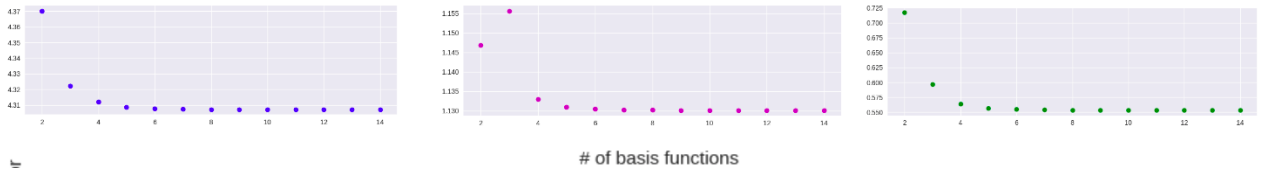


**Figure 5.33:** Error dependence on number of RBF.The right side of the figure shows error for the right movement class, the middle for the straight and the left one - for the left movement class. We can see that the smallest value of trajectory error is when we have 6 and more RBF

From Figure 5.33 we can see that the smallest error value is when we are working with 6 and more RBF, from now we will be working with 6 RBF - there is no need to use bigger number, because it do not do any influence on results, but it very affect computation time.

As mentioned before we can calculate weights on the chosen number of training data. We are a reconstructing trajectory in two ways: using the mean of weights for different trajectories and using weights, computed by using only one trajectory. Results are below in Figures 5.34, 5.35 and 5.36.

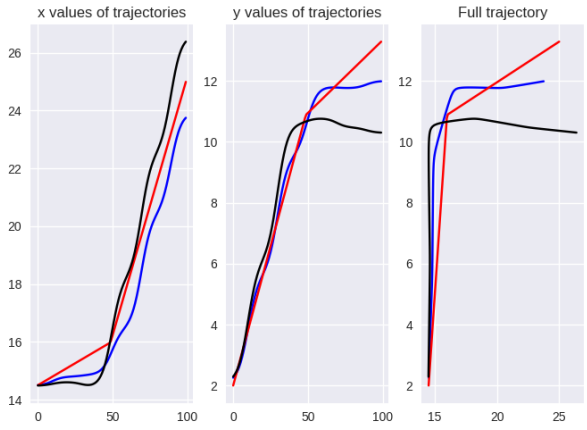


**Figure 5.34:** Reconstructed trajectories. The left side of the figure shows reconstructed x-axis, picture in the middle illustrates reconstructed y-axis, the right side of the figure shows the full reconstructed trajectory. Blue line shows reconstructed trajectory, for which reconstruction was used mean of weights of all learning data set, **black** line shows trajectory reconstructed by weights from one trajectory and red line shows original trajectory. Movement direction is right

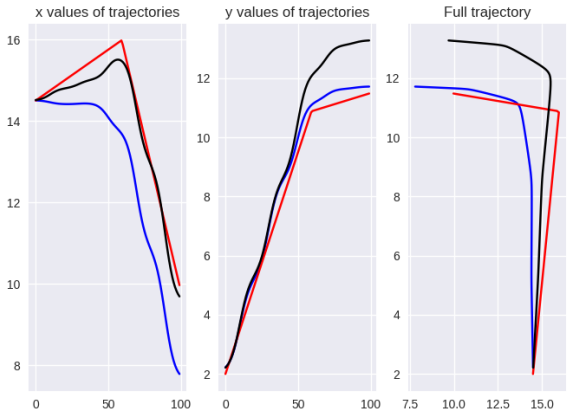


**Figure 5.35:** Reconstructed trajectories. The left side of the figure shows reconstructed x-axis, picture in the middle illustrates reconstructed y-axis, the right side of the figure shows the full reconstructed trajectory. Blue line shows reconstructed trajectory, for which reconstruction was used mean of weights of all learning data set, **black** line shows trajectory reconstructed by weights from one trajectory and red line shows original trajectory. Movement direction is left

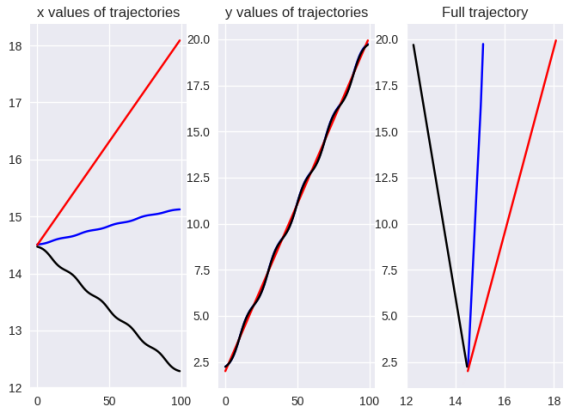


**Figure 5.36:** Reconstructed trajectories. The left side of the figure shows reconstructed x-axis, picture in the middle illustrates reconstructed y-axis, the right side of the figure shows the full reconstructed trajectory. Blue line shows reconstructed trajectory, for which reconstruction was used mean of weights of all learning data set, **black** line shows trajectory reconstructed by weights from one trajectory and red line shows original trajectory. Movement direction is straight

From Figures 5.34, 5.35 and 5.36 we can see that overall trajectory recreated quite close to the real value of the trajectory (if learning data set would be bigger, the recreated trajectory would be even more close) and all trajectories were recreated to the right movement direction.

## 5.3 Future Trajectory Prediction, Using Data, Collected In Real Time

The difference between the using pre-recorder and real time trajectories is that with pre-recorded trajectories is easier to work, since we know how trajectories looks like, we can unify them and make them to have the same amount of time steps, etc. Pre-recorded testing trajectories is very useful while creating algorithm, it allows easy and quick check of algorithm workflow. Once results are as they are expected to be, the algorithm is used on ROS and on its' visualization tool RViz. Using these tools driving simulation is happening on the real-time, receiving observed cars' position immediately after it is done - meaning, that all calculations and prediction is making depending must be done quickly and correctly.
Before starting to test prediction making algorithm we faced the problem of trajectory length. While working with pre-recorded data, we already had a full trajectory and for testing purposes, we could unify them time-steps wise. This does not work in reality, since we do not know how long driving simulation is going to happen.
We solved this problem using minimum distance which theoretically described in the chapter before. After this we tried to predict how full trajectory will look like, using ProMPs, which theoretically described also before.

### 5.3.1 Real Time Prediction Making In ROS Simulation

Working with pre-recorded data works only in simulations. In a real-life, it is very important to work with real data, which means having some level of uncertainty. We will provide the results for this experiment with the capture of images from real-time experiments.
From the Figure 5.37, it is possible to see that real-time prediction is also working. However, in order to avoid delays in calculations, it is better to use a more powerful machine which is responsible for prediction making.

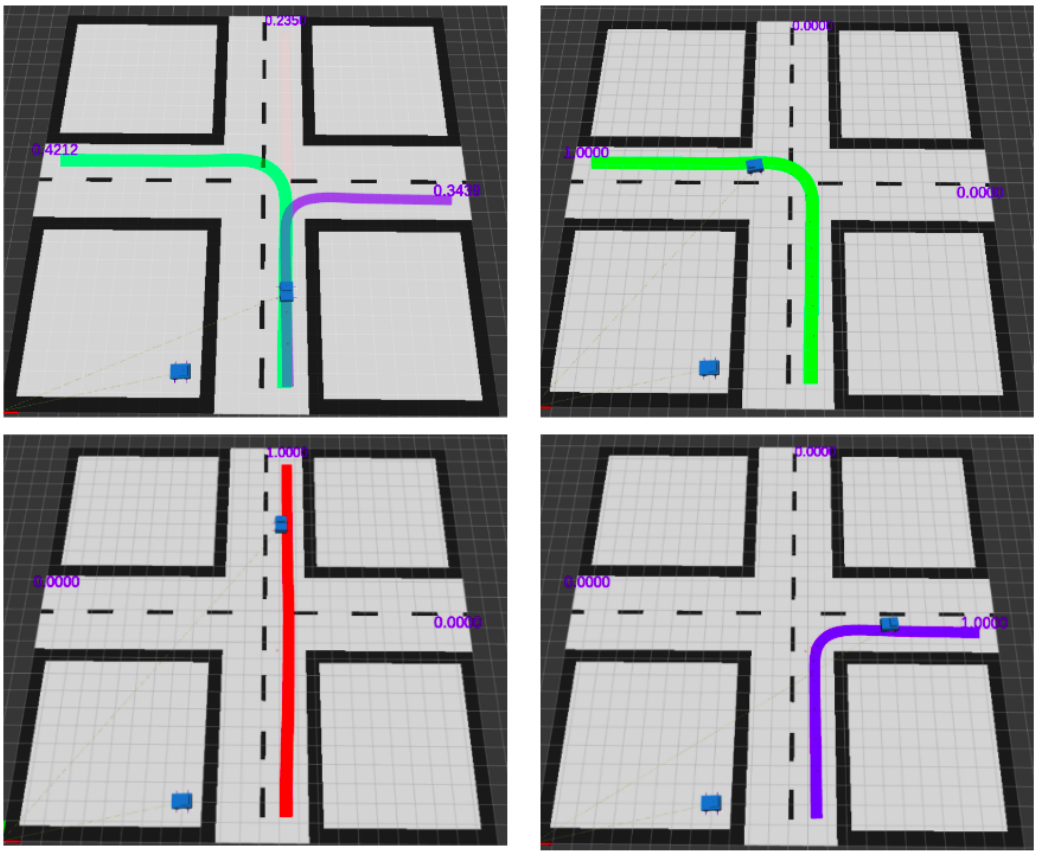


**Figure 5.37:** Few captures from proposed probabilistic intention prediction algorithm working on real-time data. The upper left side shows the prediction made while starting to drive. The upper right picture shows results while moving towards the left direction, the bottom left illustrates straight movement class and the bottom right one - right side

# 6 Security Aspects

As autonomous vehicles are getting closer to reality as private and public mean of transportation, it is natural to be interested in safety and security which these new technologies are promising. A number of security vulnerabilities and concerns related to autonomous vehicles can become a potentially deadly weapon in the future [67]. But greater concern than terrorism attacks on autonomous vehicles are the risk of controls systems being hacked in and taking control of a driving or other essential systems and in this way to put into a dangerous situation driver or passengers of the car. This can sound too non-realistic, but *white hat hackers* for years already have been showing security issues related to connected cars, demonstrating how easy is to take control and do harm over a lot of various systems even in non-automated vehicles. One of Chinese security company demonstrated that it is very easy to spoof sensors of a vehicle, causing them to sense "ghost" objects or fail to detect a real object at all [68]. Here natural question can arise, so what makes autonomous vehicles so vulnerable against malicious attacks as compared with non- or only partial autonomous cars? Authors of [2] suggesting two main reasons:

1. **The increased interaction between autonomous cars and environment.** At the moment the most communications between vehicles on the road occurs via Vehicular Ad Hoc Networks (VANETs). This type of communication allows sharing fast changing surrounding information with vehicles which are nearby communicating vehicle. This allows for other cars/drivers in the cars to be aware of what is the road situation nearby them [69]. Since technologies are improving and fully autonomous cars will be on the roads in the very near future, Vehicle to Infrastructure (V2I) and Vehicle to Internet of Things (V2IoT) communication will be more common on the roads. Due to connectivity in one network, only one "infected" vehicle can compromise the entire network if the network is not properly secured.

2. **The increased interaction between components of the system inside, so-called intra vehicular communication.** Autonomous cars have a lot of different Electronic Control Unitss (ECUs) which are connected with each other using Controller Area Network (CAN) bus. One of the biggest advantages of using CAN bus is that it is like a central unit into which a lot of different modules can be added or removed without changing the wiring architecture in the car. CAN has three main parts: **Data link layer**, which is responsible for data transferring. **High speed** physical layer and **Low speed** physical layer which is also responsible for fault toleration. In most cases, the most important control units (which has a direct impact on safety, e.g. brake or engine control module) are connected to high speed layer and others, not so security sensitive are connected to low speed layer. Not so common situation, but it is possible to have a "gateway bridge" which opens the route for selected packages from low to high (or vice versa) layers. So it is a possibility that that malicious packets are connected to the CAN bus using low speed layer and without any further checking or suspicion it can be transferred into high speed layer causing to serious consequences. Control architecture in autonomous vehicles is working in this way what all nodes get packages from CAN. Any malicious component which is connected to the internal vehicle network can snoop all communications or it can infect all other elements. In order to protect the network, every path for package moving should be protected to ensure the vehicle security. Unfortunately, it is impossible to predict all possible attacks and to foresee all vulnerable places, because there always be new strategies which will threaten the security of autonomous cars. "The development and improvement of one will always counteract and necessitate the development of another" [2].
   Another cause of CAN vulnerabilities is that all CAN packages are not authenticated before using them for communication within the system. Any element of a network can send infected element further if former did not do any validation of the package before accepting it [70]. One way to protect network infrastructure is to use packet-level authentication method. This method allows authenticating a package without having trust association of the package sender [71].

## 6.1 General Overview. Attack Taxonomy

This section will introduce the potential threats, vulnerabilities and attacks of autonomous vehicles can face. For categorization of attacks, we use way proposed in [2]. Each attack is classified based on: **Type** (or source) of the attacker, **Attack vector** (path and method which was taken to get access to the vulnerable place), **Target**, **Reason/objective/motive** of the attack and **Potential outcome**.
Figure 6.1 summaries attack taxonomy which will be described in this section.

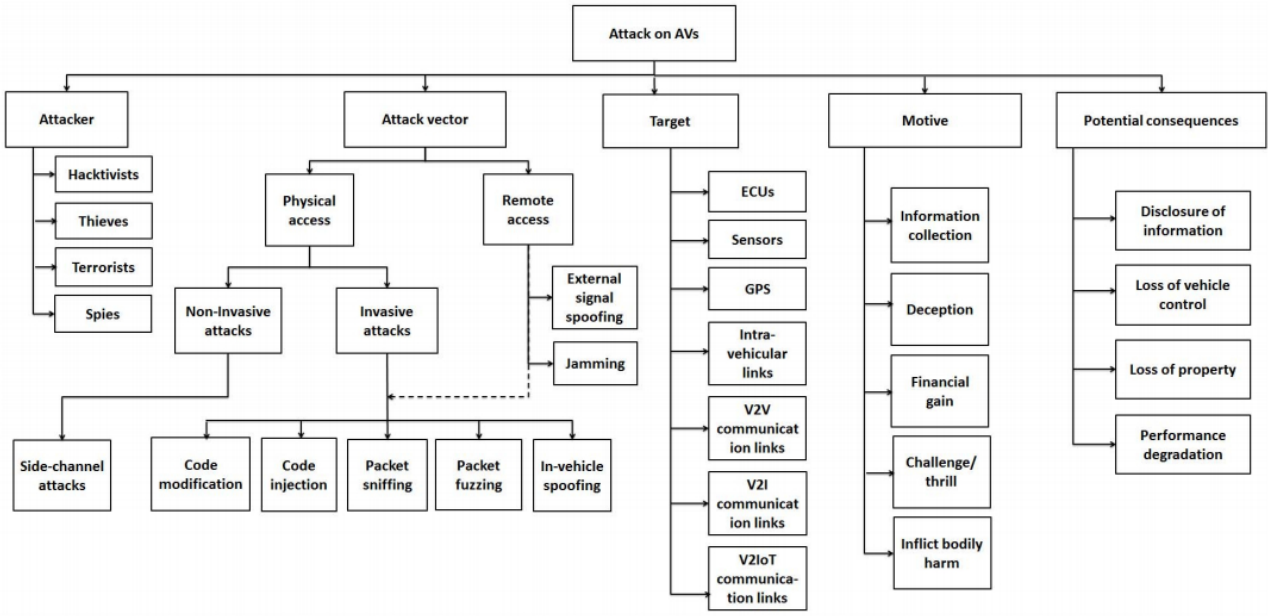


**Figure 6.1:** Autonomous Vehicle Attack Taxonomy, proposed in [2]

### 6.1.1 Attacker

It is a source of the attack. Usually, when system face an attack it tries to mitigate that immediately. One of the steps of mitigation should be an identification of attack source. Having this information it is possible not only to prevent future attacks, but also understand why the attack was implemented in the first place.

### 6.1.2 Attack Vector

It is a way how the attacker got access to the system he is targeting. It is also an enabler for an adversary to exploit the targeted system. Attack vectors can briefly be categorized to *physical* and *remote* access.

#### Physical Access

These attacks are classified into invasive and non-invasive attacks.

1. **Non-invasive Attacks**: these attacks have no physical contact with the car, usually, embedded devices are used for attack implementation. However, being relatively close to the targeted vehicle is necessary in order attack to work. Example for non-invasive attacks, can be *side-channel attacks*. These kind of attacks usually ends with leakage of useful information about transmitted data or about internal working paths within the system. The most common defense against this attack is employing asynchronous information processing units or/and "shielding" mechanisms.

2. **Invasive Attacks**: the main difference as compared to non-invasive attacks is that invasive attacks include a physical connection to the targeted system. The result of this attack can be the network security and ECUs can be compromised. Potential way how adversaries can connect to the car and gain access to its ECUs is On-Board Diagnostics-II (OBD-II) port which is usually used for car diagnostic. Another way to reach a car is wireless remote access, this is possible when an autonomous car is connected to the critical infrastructure, e.g. someone in the car connects a smartphone for entertainment reasons, and in this way, all internal system can be exposed to external people or networks. There are different types of invasive attacks which are discussed below.

    a) *Code Modification*:this attack may happen when attacker change the code with malicious modifications in order to compromise the system. This may be achieved by connecting OBD-II scanner to the car. As mentioned before this is a tool for vehicle diagnostic, meaning that it is widely available for anyone who wants to buy it. This attack can be avoided by ensuring that all connection to the car is password protected, which enable only certified people to connect and modify codes of the car.

b) *Code Injection*: this is a very similar attack to the previous one. In this case, after connecting to the car, the attacker can inject new (most likely malicious) code instead of modifying the old one. Code injections can be made not only by an attacker, the owner of the vehicle or person who is checking a car can also inject a new code, hoping to improve the performance of a vehicle. When injected codes are non-compliant with car's components or when new codes are not proved by authorities, problems can appear. One of the ways to avoid code injections is a usage of the intrusion detection system or/and usage of privileged access, which allows only authorized people to connect to the car, owner excluded.

c) *Packet Sniffing*: also known as package analyzing. For this attack computer program or hardware, called sniffer (or analyzer) is used. A packet sniffer can see details between any communication node. The tool is very useful when network related problems need to be diagnosed. But the tool can be used for malicious purposes as well: an attacker can use sniffers for collecting unprotected information, for eavesdropping or capturing packages for following replay attack. Defenses for this attack can be various encryption techniques for confidentiality in packages protection, as well as usage of real-time packages send/received update messages.

d) *Packet Fuzzing*: this technique usually is used during security testing procedures, when invalid data is sent to the system/element, expecting to get receive some error or fault conditions, which allows exploiting weak places in the system and security loopholes. While testing the system, fuzzing helps to detect problems and utilize them in a further stage. When this technique is used by adversaries, the received information is used to get into the system and do any possible harm. Protection against fuzzing is to fix all errors and security loopholes immediately after its exposure.

e) *In-Vehicle Spoofing*: is a situation in which an attacker, using some software tools, pretending to be another person by falsifying data. In this way, adversary gains an illegitimate advantage. In order to attack be successful adversary needs to overcome security mechanisms (if exists) to replace original elements with spoofing devices. Usual defenses for this is into autonomous car network include reply attack resistance techniques and fingerprinting module to be able to differentiate between original and current module in the vehicle system [72].

### Remote Access

Since wireless connection to external sensors, such as cameras, LiDAR, Radio Detection and Ranging (RaDAR), GPS are getting more and more popular and useful while driving, attackers can also use remote access as method to attack systems of autonomous vehicles.

1. **External Signal Spoofing**: one example for this type of attack is GPS spoofing. This attack is possible because GPS using a wireless connection. During the attack GPS receiver is deceived by sending the wrong signal to GPS from another device. The incorrect signal may be similar to real GPS signal or can be captured before and replayed at a certain time. An attacker can trick GPS receiver to accept and recognize only fake signal by gradually increasing power strength of the wrong signal until it eventually replaces the original signal. As soon as the attacker gains control of GPS device in an autonomous car, he can send false GPS information and lead car in the wrong direction and destination. This attack was not tested on autonomous cars, but it was successfully tested on GPS devices in Unmanned Aerial Vehicle (UAV) and yachts [73, 74].
   GPS devices are not the only target in autonomous cars. Vehicles contain numerous sensors, which can be attacked with spoofing. One of these devices could be visual sensors like cameras, LiDARs. Similar to the previously described attack was made on LiDAR sensor in [75]. Sending fake signals to the device in a range between 20 and 250 meters from a car (sensor) a lot of non-existing obstacles can be detected as well as existing obstacles on the road can be missed. The way to protect the system against spoofing attacks is to ensure more security, not to accept any signals and information without checking the authenticity and integrity of a signal sending device. To ensure that correct information is coming to LiDAR additional information sources can be used and perform information checking between two different devices. If this cross-checks matches and succeeds information is more likely to be correct than in a case when information is not matching between different sources.

2. **Jamming**: these attacks are against wireless or external sensors for vision and due to that, the authorized communication might be destroyed. The most sensitive devices for jamming attacks are LiDAR, RaDAR, various cameras. Jamming devices can block sensors for receiving correct data. Authors of [75] used jamming to blind cameras of autonomous vehicles to hide objects on the road and make map "cleaner" as it is. There are ways to protect sensors against this attack using removable near infrared-cut filter to the camera, however, this method is working only in the day time. Another measure is photo-chromic cameras' lenses, which can filter out specific types of light.

### 6.1.3 Attack Target

The target component of the car is usually depends on motive/object of attack and attacker's intention. If the attacker wants to track a path which car is moving, the target element can very likely be camera and/or RaDAR or LiDAR , since they are vision elements of the car and having information from these sensors it is easy to see and follow the path, car took. If attacker targeting traffic optimization and/or passengers safety, as a target VANET can be used, ect.

### 6.1.4 Attack Motive

Various motives for attacks are possible, better we understand it, it is more likely to protect all control systems and ensure safe driving. One of possible reasons for attacking can be deception - this is a spreading a false information, hoping to effect behavior of other, this attack may lead to hazardous situations on the traffic/roads. Another motive can be economical - to get a financial gain. One more reason can be, due the any reasons, cause severe to passengers of the car, etc.

### 6.1.5 (Potential) Consequences

Various consensuses are possible if any of these attacks against autonomous vehicles will be successful, such as some (important) control functions might to fail or be sabotaged, data leakage (e.g. vehicle movement information) is very possible outcome [2, 76]. The "health" of the system is very likely will be affected after successful attacks, which can lead to passengers of the car health issues.

Another important problem which must to me mentioned while talking about security issues on autonomous cars and is not mentioned in attack taxonomy yet is **privacy**. An attacker can arrange an attack where he follows car movement, times and makes a very detailed profile about car owner and with this information can do various things: from robbing the house of the victim while he is not at home, to selling this information to someone else, combine different attacks to do the biggest damage attacker want (or is able) to arrange.

## 6.2 General Overview. Defense against Attacks Taxonomy

As mentioned earlier it is very hard to predict what is going to happen to prevent yourself against attacks. However, with current knowledge about system and known potential vulnerabilities, it is possible to make some kind of predictions and develop network architectures and working protocols which are not so vulnerable. Various literature surveys propose the main 4 types of defenses for autonomous vehicles: **Preventive**, **Passive**, **Active** and **Collaborative** defenses [76].
This classification and different ways of protection ensure that the system is secure and resilient for different attacks. Figure 6.2 summaries defense against attacks taxonomy described in this section.

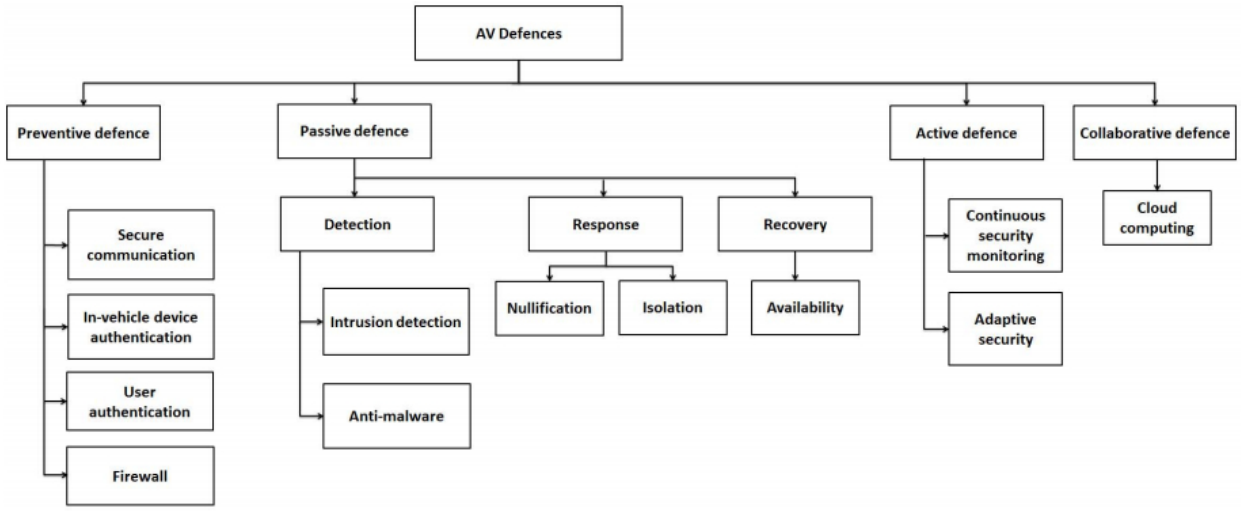


**Figure 6.2:** Autonomous Vehicle Defense Taxonomy, proposed in [2]

### 6.2.1 Preventive Defense

This type of defense mechanisms mainly focuses on protecting a system from attack before it starts or finishes with success. Preventive defense is mainly focusing on normal working conditions and not during the attack and does not solve any issues with "after attack" scenarios.

1. **Secure Communication**: for secure communication data encryption is basic and crucial. Using encryption content and confidentiality of messages are assured. Encryption scheme can also help with the identification of sender/received of sent data. To know the integrity and authentication of another side of communication is very important for secure communication.

2. **In-Vehicle Device Authentication**: controllers used in the car can be completely trusted if they have a manufacturer certificate which gives information such us controller identifier, public key, information about authorized carryout, etc. If information, provided in certificate matches with information which car itself contains, then authentication process is successful and car safely can use all information coming from that particular controller.

3. **User Authentication**: sometimes, in order to have more protection, user authentication is used. To make sure that the right person has access to the car (e.g. doors opening, starting the car, etc.) additional biometric information can be used.

4. **Firewall**: it is an additional tool, which always can be used. Firewalls check all incoming and outgoing data traffic based on rules which user/authenticated people can define. Firewalls also can be very helpful in communication with a trusted and not-trusted environment: this is very important while communicating with different objects in vehicles' networks.

### 6.2.2 Active Defense

This is one of the advanced and determined defense techniques. Different approached described below.

1. **Continuous Security Monitoring**: autonomous vehicles belong to critical infrastructure and it is essential that security of these systems should not be compromised. It must have (near) real-time solutions for checking and/or restoring healthy driving conditions in the car. Non-stop and continuous monitoring and "snapshots" of all running systems are required to be available for security checking at any time.

2. **Adaptive Security**: Nowadays the most critical systems are fast changing or refer to infrastructure with a fast pace, it is necessary to model and use defense mechanisms which are dynamic. So-called adaptive reconfigurations for target can be used for ensuring better control and balance during attack. In order to prepare it self for future attacks defense mechanisms should be able to analyze past and current attacks and use self learning mechanisms to predict what may happen in the future and adapt defense mechanisms for new forms of attack in the future.

### 6.2.3 Passive Defense

Passive defense measures are taken to minimize damage caused by an attacker without having the intention of taking initiative. A passive defense can be an additional level of protection (not the main one). As compared to active defense, the passive defense does not require any analysis from human.

1. **Attack Detection**:
    a) *Intrusion Detection*: To detect physical threats to the car is much easier than to see attacks against system operations. However, there are various models for Intrusion Detection System (IDS) which can be used for autonomous vehicles. Authors of [77] proposed and tested IDS models using various computational simulation scenarios.
    b) *Anti-Malware*: these systems need to be able to protect from harmful attempts to penetrate into the main system.

2. **Attack Response**:
    a) *Nullification*: when an attack is recognized by system nullification can be used. This defense mechanism can neutralize an attack using cyber/electronic capabilities, e.g. GPS signal anti-jamming technologies [2].

b) *Isolation*: it helps vehicles to isolate themselves from other cars during an attack. Self-isolation also prevents ECUs re-programming while the car is running. When a car is attacked ideally it not only should isolate itself but also inform vehicles around about attack in order other cars could take some actions to defend itself against attack [2].

3. **Attack Recovery**:

a) *Availability*: this feature one of the most important in all types of systems. In the context of autonomous cars, availability is very important when talking about safety inside and outside of the vehicle. In order to ensure safety and have good fault toleration within the system in autonomous cars and to ensure quick recovery after attacks, availability must be ensured in the system.

This section provided a brief summary of vulnerabilities, attacks and defense mechanism in term of security of autonomous vehicles. One of the most important things considering security is don't to stop looking for potential vulnerabilities and ways to protect the car and all network in case of attack and/or to recover as fast as possible if an attack happened. Regardless of the effectiveness of current methods, it is necessary to think about new defense mechanisms for real-time operations of the vehicle.

## 6.3 Attacks Related To Intention and Movement Prediction Making

[75] proposed the scheme from three phases which described all flow in autonomous cars: Sence, Understand and Act. The first phase contains all sensors and raw data collected from them, the raw data is proceeded and represented in the second phase and finally, having all information, in the third phase decision and action can take a place. All these phases are equally important, however, all this algorithm cannot fully functioning with poor (or fake) sensors data.
Our proposed algorithm as observation uses only position information, due to that attacks on GPS are presented. To improve prediction making process data from another sensor might be included in the algorithm. For environment perception, camera and LiDAR are one of the most important sensors in the car. Further in this section will be discussed the most common and dangerous security attacks: visual sensors (on camera and LiDAR) attacks and common privacy issues.

### 6.3.1 Attacks on GPS

GPS receives the signal from satellites collection which flies in Earth orbit. The data signal from satellites contains the payload of time-stamp, the location of the satellite. Speed of radio waves are close to the speed of light clients (cars) use the time-stamp to measure the distance of the satellite. After all data alterations signal shows latitudinal and longitudinal coordinates, giving a precise location of the object as compared to satellites.
At the moment attacks on GPS is not a concern - they explored more in the laboratories at Universities rather than by criminals. However, there is a possibility to arrange attacks against GPS and this is only a matter of time when attacks against GPS will become a real life problem. To attack GPS two main attacks exist: jamming and spoofing. Signal Jamming is a noise passing over the GPS frequency a band which results as in a Denial-of-Service for the user. While signal spoofing is false data transaction to GPS.

#### GPS Jamming and Spoofing

[74] describes a spoofing attack on GPS in a yacht. Attack was arranged strictly on academic purposes, however, it shows that it is very easy to spoof GPS. As mentioned before, GPS signals are received from satellites. However, researchers in the University of Texas showed that it is possible to used spoofing device and create a fake GPS signal on Earth. "Attackers" were sitting inside the targeted object (yacht) and started to spoof fake GPS signal which was a bit stronger than the real GPS signal, coming from the satellite, using their spoofing device (which is easy to buy online). The attack started when navigation system of yacht caught a fake signal. "Attackers" changed fake signal only a few degrees, but that made all systems in the yacht to think that it is not in its course (however it was). In these situations, captain, just adjust course according to GPS readings to be on the right course. What in reality means that yacht is few degrees out of its real trajectory. Few degrees can look not too many and nothing can happen, but these alterations can cause a lot of accidents since location precision of any means of transportation is a very important variable. In the end, by changing GPS readings the destination of the trip can end in a completely different direction than expected.
However, GPS spoofing cannot take control of the driver. It just shows the wrong location to go. The driver is the one who physically modifies the path, trusting readings from GPS.

**What is the difference between GPS spoofing and jamming?:** GPS spoofing can trick the navigation system with false data, whereas jamming can disable the GPS system entirely. An attacker can jam any noise signal with the same frequency as GPS signal is and at the end, GPS device cannot differentiate which signal is real, which is not, since it got a mix of everything. In this case, GPS system is not usable at that moment.
To protect the system from jamming is possible by using anti-jamming devices or nullification, which was described before. To avoid spoofing attacks it is useful to authenticate the sender of the signal since now majority of GPS systems blindly believe that GPS signals come from satellites and do not do any sender checking.

As mentioned before, in this thesis we are not using any data from sensors, but to improve prediction making in a real life it is very beneficial to use additional information with the position. Due to this reason more serious attacks on visual sensors, which are the most important sensors in environment perceptions, are described.

### 6.3.2 Visual Sensors Attacks

In reality, autonomous cars are equipped with various sensors which measure different physical properties (sound, light, distance, radio frequency, etc.), GPS, LiDAR, RaDAR systems and of course maps. To ensure sensors' resilience against various attacks is one of the main challenges. And yes, any of possible attacks against sensors can cause a decision, which can end up as an accident and in the worst case can lead to fatalities. E.g. attacks on camera can cause misunderstanding of traffic signs, leading to unsafe driving conditions with additional danger to passengers in the car and/or other road participants. LiDAR can be fooled and observe non-existing obstacle on the road and start braking, or LiDAR can be triggered to not notice real obstacle and go directly to it. These are only a few scenarios, but all of them can make a huge impact on traffic and its' participants [78].

#### Attacks on LiDAR

For proper car guidance, which ensures safety on the road proper visualisation on an object on and around the road is necessary. To have good quality data with image-based sensors good weather conditions is required (typically they cannot give proper geometry information under poor weather conditions, e.g. when is raining or when it is not enough light). Laser technologies here had a lot of advantages and LiDAR was introduced. LiDAR working principle is based on laser scanning, and it can accurately capture all types of geometry on the road and to detect a very wide range of objects. However, a very detailed grouping of objects is still a hard task for LiDAR. Recognition of traffic signs or lanes on the road is still cameras' task.
The brief working principal of LiDAR is to notice objects which are giving a reflection of a signal sent by LiDAR. If a sent signal does not come back to the device (it can happen because of transparency of the object, absorption, range limit, when in short range there is no object, etc.), LiDAR thinks that there is no obstacle on the road. As LiDAR performs a very important role in environment perception for autonomous cars and uses simple methods for object noticing, such as light pulses, it is one of the most obvious targets on the autonomous cars. The easiest way to attack LiDAR is to create "noise" and generate (or hide) objects. This section will provide a description on already existing and tested replay and spoofing [75] attacks on LiDAR

**Signal Relaying Attacks**
Relaying the signal is an expansion of replay attack, which goal is to relay on the signal which was sent before the attack from another position (signal sent from LiDAR is recorded and then (repeatedly) sent from a different location). This allows creating fake echoes and due to that obstacles may appear closer/further than they truly are.
To perform an attack, only two transceivers, one with a photodetector, sensitive to wavelength LiDAR is operating on and one with the laser are necessary. The output of transceiver with photodetector is a voltage signal, which corresponds to the signal intensity which was sent from LiDAR (to be able to see the signal, oscilloscope needs to be connected to the transceiver, but it is not necessary for performing an attack). The output of the first transceiver is sent to another one transceiver (which has a laser in it) to emit a pulse in as its output.
In [75] experiments made when transceivers were in the one-meter distance from each other and in front of LiDAR. But this position is not only one which must be used for making this attack work: it can work very well when transmitters are behind the LiDAR with a bigger distance between each other. A relay attack on LiDAR is most likely to be performed from the roadside, where an attacker can easily receive signals sent by LiDAR on a car, to record them and relay them again from the different location.
Results of an attack, which were noticed by performing it was that before an attack LiDAR only noticed objects detected

in short distance (around 1 meter), during the attack range of noticing objects had grown to 20-50 meters - during the attack, it was noticed that LiDAR emits not encoded pulses, and signals can be recorded, replayed and relayed to create fake echoes. Due to this movement planning and decision making can be affected, together making an impact on people and traffic safety.
Note, that to perform the relay attack is relatively easy and cheap, due to that it is widely used for attacking cars in general, not only for attacks on LiDAR.

#### Signal Spoofing Attacks

While signal relay attack can create echoes, this paragraph will show that it is easy to create fake objects using signal spoofing attacks on LiDAR using the original signal released by LiDAR to "replay" objects on the road and control their location. A signal, sent from LiDAR, travels approximately 200 meters back and forth in about 1.3 $\mu$s. Meaning that LiDAR should listen at least this period for incoming sent signal reflections (depending on LiDAR type time for a signal travelling might vary a little bit). In order to make attack successfull, the fake signal must arrive to LiDAR in this small time window, if the signal will come back to LiDAR after this time, it won't be noticeable, that's why attacker needs to know when to release fake signal. Working principle of LiDAR is that longer time it takes for a signal to come back, the further object is, due to that the attacker can "control" location of obstacle by delaying the original LiDAR signal before relaying it. [75] demonstrates an attack, where LiDAR receives the fake signal after the first echo of the original signal is received. This allows tricking LiDAR to think that obstacle is further away since the signal traveled back longer. In order to make the attack work attacker needs to have a transceiver and two pulse generators (they are needed to generate a fake signal, which is sent back to LiDAR). The output of transceiver is connected with the input of one pulse generator (this generator delays its output). The output of this pulse generator is connected with the input of the second (pulse) generator. A defined number of square-wave pulses are generated when the second pulse generator is triggered. These newly generated wave pulses are sent to transceiver. All variables (time for delay in the first pulse generator, number of generated pulses and copies of it, as well as pulse width and its period that is received from the second generator) for this attack can be controlled, by doing this it is possible to "create" all kind of obstacles with all kind of distance between the new object and car. Results of the attack showed that objects which are in around 50 meters distance, usually are noticed within the second echo and by tuning delays, it is very easy to make them closer or further. The first pulse generator can be configured to send multiple pulses to the second pulse generator when it receives a signal from LiDAR. Due to that, it is possible to sent multiple fake signals in sequence back to the LiDAR. The resulting in noticing multiple objects in the desired distance between each other (the first copy will appear in the second echo, the third copy in the third echo and so on until time for LiDAR listening will finish).
The same technique can be used for hiding objects and make LiDAR not see any obstacles, just in this case the pulse generator generates a copy of the signal sent in a "clean" rode.

#### Countermeasures

The most of countermeasures against LiDAR attack can be implemented in software. Usually, no modification of hardware is necessary, although to make devices more secure manufacturers can improve hardware implementation schemes by adding more secure (or additional) devices. Further in this paragraph countermeasures, proposed by [75], which do not require changing of hardware, will be discussed.

1. **Redundancy**: [79] demonstrated that it is possible to arrange a successful spoofing attack using different wavelengths than proposed before (as long as wavelengths are not overlapping). However, some wavelengths have some disadvantages as compared with others, combining multiple various wavelengths on the same LiDAR operation makes it much difficult to attack them at the same attack.

2. **Random probing**: LiDAR is sending signals with fixed intervals (which depend on LiDAR scanning speed and rotation speed of mirrors inside the LiDAR) between each of them. To make an attack successful, the attacker needs to know exactly when a fake signal must be sent back, for that he needs to synchronize his system with LiDAR operating intervals. One way to avoid attack could be varying these intervals in a not predictable way, however, it can be problematic for LiDAR rotation, because they need to keep constant rotation pace to know at which angle they are operating at the current moment.
   Another way to protect the car from attacks (or more precise to be aware of them) is to randomly skip signals sent from LiDAR. When LiDAR skips sending a signal, he still can listen, if the signal is coming back then it could be an indication that the car is under attack. If the frequency of sending signals is 50 Hz, skipping a few signals won't affect the quality of the LiDAR results, especially if the object is close by, however, this method can affect the quality of LiDAR results if the operating frequency is different.

3. **Shorten the pulse period**: As mentioned before, LiDAR signal usually needs about 1.3 $\mu$s to travel back and forth of 200 meters. This time window is also available for attacker operation. If the signal period would be smaller,

it would give less time for attack as well. If this defense technology is used, it is necessary to be aware that if together with a signal period, the maximum range of going back and forth should be reduced as well: if the signal period is reduced to 0.65 $\mu$s, the signal traveling range should be decreased to 100 meters too.

## Attacks on Cameras

The camera is one of the very important devices in an autonomous car, it can detect traffic signs/lights, various objects on the road, etc. However, there are various ways of how the camera can be attacked: traffic sign recognition can be tricked by adding fake traffic signals at a false location, they also can be visually hidden by surrounding them with shapes which are not considered in the recognition algorithms. As mentioned before, any object recognition using a camera has its own drawbacks due to computation power or/and quality of view, also the camera very depends on the day time (during the night object recognition is more complicated), this opens a lot of different ways to attack the device itself, like attack auto-focus or light sensitivity on camera.
Further paragraphs describe attacks whose goal is to hide objects and trick auto-controls of the car. To prove that attack is possible, an attacker only needs a laser pointer or Light Emitting Diode (LED) light. During the experiments, authors of [75] uses tonal distribution (on grayscale value), with a total of 256 bins.

#### Blinding the Camera

As the name of attack hints, the goal of the attack is partially or fully blind the camera. Failing to recognize various objects on the road, such as traffic lights or/and signs car really affect the safety of people in the car and other traffic users.
The camera is considered to be blinded when a camera cannot tune the auto exposure or get it down anymore. When this happened, the light cannot be shadowed and image from camera becomes overexposed. The main 3 elements exist, which can make an attack less or more effective: **environment light** - when the camera is in a bright environment, more additional light is needed to raise the light in order to reach camera blinding point; **light source, which is used for blinding** and **the distance between light source and camera**. To test multiple attack scenarios tests were made in bright and dark environments and with different distances between the camera and light source (0.5 m, 1 m, 1.5 m and 2m). The thing which must be considered: when the distance between the light source and camera is bigger, more light sources are needed. Experiment as held when the light source was in front of the camera. The results of experiments showed that laser with 650 nm diode point is the most effective in a short-range environment. When laser if off, camera obviously can see the background and all view in its visibility range, when a light source is on, the background is not visible and the camera is partially blinded: light source shifted camera's tonal distribution. In the brgiht environment, the most effective light sources are the 650 nm laser, while in the dark, the 940 nm laser has the most influence. Even though experiments were not successful in achieving the full blindness, these light sources still are good enough for attack since even with partial blindness images from camera is not readable. More detailed results can be found in [75].

#### Countermeasures

Some methods exist to protect cameras from being attacked. Unfortunately, most countermeasures require serious hardware changes and this increases not only price range, but the size of the device as well, which can cause some problems for space limitation in the automotive environment.

1. **Redundancy**: As usual, multiple cameras which capture the same image could make a big challenge to an attacker, since he needs to blind multiple cameras at the same time. Using multiple cameras might not protect against well-prepared attacks which are using a very strong light source but for usual size attack, it would be enough. However, to put more cameras on the car requires more space on the car nominated only for cameras, it also requires more calibration, because of images with overlapping places can misalign between each other.

2. **Optics and materials**: It is always possible to add safer hardware parts to be able to ensure more resilience to attacks. Removeable near-infrared-cut filters are already available and used on security cameras, it can filter infrared lights on request. During the day time, it gives better quality pictures and during the night filter is removed an only camera's infrared light is used for night vision. Since the filter is useful only during the day time, it can protect cameras against attacks also only during the day.
   Another way to protect cameras against attacks is to use photochromic lenses, they can be set to filter only specific types of lights. As an example for photochromic lenses could be darkening glasses, which would be useful for sunlight. Depending on the type of lenses, it identifies which type of light it can (or will) filter. The advantage of these lenses is depending on the type of materials it could be that they do not affect the quality of the image in a low-light environment.

### 6.3.3 Slight Object, Captured with the Camera, Modification and How Does that Affect Images Recognition

To operate safely and correctly autonomous cars, as other robot-based systems have "to see" the world. For this reason, a lot of cameras are used, without them, algorithms of autonomous cars cannot fully function. The section above described some attacks on the camera itself, this section will talk about modification on street signs which can trick visual recognition algorithms in the car. As stated before to see the world pictures from cameras are used. But usually, between the images which camera is taking are some gaps, which hide some information. To solve this so-called black box of machine learning algorithms are used, these algorithms are trying to interpret some common patterns between pictures into something algorithms are familiar to. Before using these algorithms in real life they must be trained before. Usually, in this process a lot of pictures of the same thing with some differences are showing, then the algorithm is checked if it can recognise a picture of the same thing which is not in the data set which with algorithm was trained.
This training method works quite good, but it is more complicated than it can look: algorithms do not look common feature in the manner of "look for a red sign with STOP on it" (for stop sign). They are searching for features which are not so easily recognizable to human eyes. It can look a bit not understandable for human, but this machine learning algorithm recognition model works because there is a fundamental difference between how the human brain works and how the world is interpreted with artificial intelligence. Meaning that any visible changes (does not matter how big or small they are) can be understandable by human and by the algorithm in a completely different way. Even though it does not look like slight modifications on images require complicated analysis and various image manipulations to recognize the correct object. Authors of [3] showed that it is possible to fool machine learning algorithms and image recognition by introducing very small changes on the physical road sign: a bit of paint or stickers on a sign can create a lot of troubles in recognition e.g. stop sign instead of thinking that it is speed regulation sign.
[3] showed that combining the picture with the adversarial image while processing captured photo can cause vision system to recognize something completely different, an example can be seen in Figure 6.3. This can cause a lot of damage to all traffic participants.

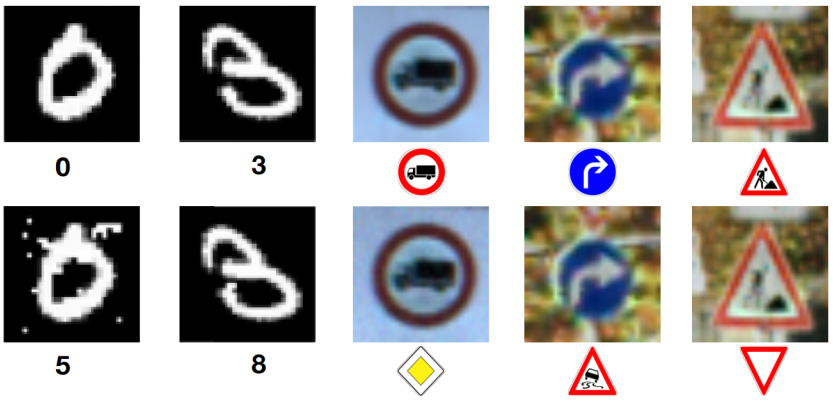


**Figure 6.3:** The upper line of pictures shows the correctly recognized images (pictures had no adversarial information combined with original image). The bottom line of pictures shows captured pictures, combined with adversarial information, and what is recognized with the same algorithm [3]

Attacks, introduced in [3] are effective, but it is much harder to make it a real life rather than in the laboratory. The advertiser usually does not have direct access to control what is coming as an input to a vision recognition algorithm. As well, usually there are a bunch of other pictures of the same place in different directions and angles, so different algorithms can compare the pictures of the same place in different situations. And the last thing which does not usually work in the real world is that adversarial images contain the same features in all image not separating traffic sign or background.
The difference of method proposed in [4] is based on changing the sign physically - to alter signs in the way that visual recognition algorithms would be not able to correctly recognize the sign. For that author came with some "sign damaging" techniques: "subtle fading, camouflage graffiti, and camouflage art". Figure 6.4 shows how perturbed signs, used in experiments, look like.


**Figure 6.4:** Small perturbations on signs resulted in misclassification of stop signs and think that it is speed limit 45 sign. The sign to turn right was recognized as a stop sign [4]

There are different and easier ways to "damage" signs to mess up with visual recognition algorithms. It is possible to add some stickers or graffiti on signs as in Figure 6.5.


**Figure 6.5:** Camouflage graffiti and art stickers caused visual recognition algorithms to recognize stop sign as speed limit 45 [4]

Due to being smaller, stickers have a visibly smaller zone, their created perturbations have a more significant impact on sign recognition but less visible by the human eye, and it worked as well. According to the authors of [4]: "*The Stop sign is misclassified into our target class of Speed Limit* 45 *in* 100% *of the images taken according to our evaluation methodology. For the Right Turn sign. . . Our attack reports a* 100% *success rate for misclassification with* 66.67% *of the images classified as a Stop sign and* 33.7% *of the images classified as an Added Lane sign. [The camouflage graffiti] attack succeeds in causing* 73.33% *of the images to be misclassified. In [the camouflage abstract art attack], we achieve a* 100% *misclassification rate into our target class*".
Authors of [4] for algorithm training used publicaly available labeled data set. They assumed that the attacker won't be possible to play around with training data, but he can send images into an algorithm and see what results are coming out, to be able to see how algorithm recognize signs. And at the end authors took the "normal" image of the sign which is going to be attacked, gave it to the algorithm and received an adversarial image for result. It is probably a good assumption to make those classifiers which are used for autonomous vehicles have more sophisticated and robust than those which were used by authors. And it would be naive to think that hackers won't ever figure it out how to walk around even the most sophisticated and robust classifier ever. Probably the best defense against these attacks would be to use a multi-modal system for road sign detection as they are using it for obstacle detections - it is not very wise to trust only one sensor for these crucial tasks [80].

### 6.3.4 Privacy Issues

While autonomous cars might sound very convenient and create new forms of accessibility, we do need to remember that most of this comfort comes at a very big cost of privacy. To be able to ensure safety rides and all comfortability most of the people are expecting autonomous cars has a number of various sensors, which are proceeding a lot of data. Basic sensors of the car is visualized in Figure 6.6.

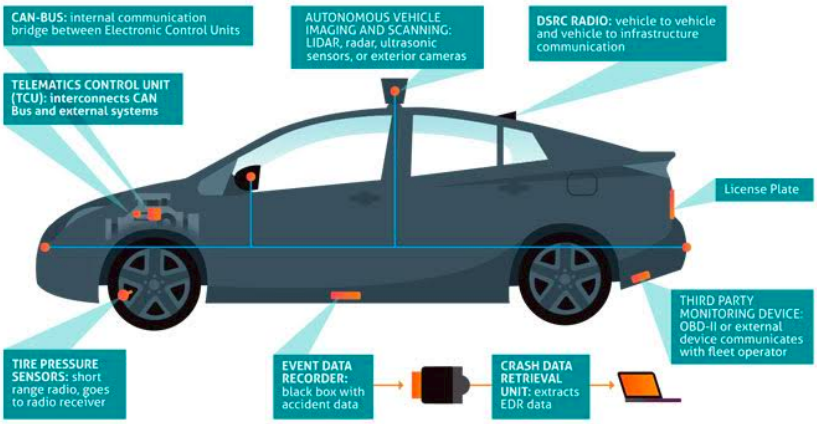


**Figure 6.6:** The Basic Data-Generating Devices and Flows in Autonomous Cars [5]

Article [81] provides a very clear scenario of how "Automated vehicles will learn everything about you—and influence your behavior in ways you might not even realize". Most of the people in the morning go at work, so considering the time of the day, when an owner of the car gets into it, an autonomous vehicle will ask (or suggest) about driving at work first. Later, during the ride to work, car's owner want to take a coffee and car can suggest the coffee store or inform about sales and other offers which are happening in the stores which are on the route to work. On first sight, it might look awesome, but on the other hand to make these suggestions car needs to have some sensitive information about car's owner life and habits, as well as potentially be influenced to announce sponsored content. When a car makes an offer to go to work, it knows that at this particular time, the car is used to go to work. When a car gives you information about sales and coffee place, the order of offers coming out is based on how much business paid to advertise this offer (more expensive advertisement is, faster it is announced). Even though a lot of people can not see any problems with it, however, it can hide two big problems. First, car for obvious reasons always retains information where the car is and this can be very valuable knowledge for attackers/hacker. By knowing the place where and when the owner of the car is working, the attacker can make an assumption about the financial status of the car owner and knowing the time he is not at home, an attacker can easily break into his house and rob it. Additionally, during the robbery, an attacker can still track the car location and whenever car starts to move from current location, it can indicate that owner os potentially coming back to home and it is about time to leave. The second problem is that accepting the first location car suggested could give the car's owner personal data to marketers for highly specific marketing purposes without him/her even knowing this.
To consider these aspects is very important while thinking about the adaptation of autonomous cars, however, all legislation on autonomous cars so far are very vague when it is related to the aspect of privacy. So far only plan to keep the car owner informed how and where his data will be used is defined [82]. This is a start, but still privacy issue not even close to being clear. A lot of people agree that it is crucial to regulate at least "minimum acceptable level of security" when it comes to data from autonomous cars, as there are minimum safety standards for the cars nowadays. We already have a very high number of issues for privacy which is related to the usage of smart technologies and it would be wise to think that the number of issues only grow with autonomous cars. Laws for privacy as always difficult to define, however, these questions are crucial for a secure transition to the safe world with autonomous cars.
Personal privacy is not the one concern which needs to be addressed. The case in 2016 when FBI was trying to force Apple to give access to personal iPhone of person who was suspected to be a shooter in San Bernardino attacks [83], raised a legal issue: in which precedent manufacturer can (or need) to give personal user data to law enforcement. In the case, mentioned in [83], the phone was an Apple product, but the data stored in the phone was the individual property of the phone owner. Could the law enforcement/government force the manufacturer of the autonomous vehicle to decrypt the personal data and give all information where someone was and where potentially could go? Again, on the first sight, it can look very innocent for cars' manufacturer to reveal the data of "dangerous" person, but this as well could affect the general consumer. When the manufacturer create a "back doors" which can give access to data of the user, just for "in case" scenario, these "doors" can be used not only by the manufacturer, it can be used by a malicious hacker to access a

data and gain an important information which could be used to harm innocent autonomous car owner. In theory, to be able to protect consumers data, a manufacturer should ensure a security level of a device/car that even they could not get access to it in any cases: yes, this can save some criminals from being caught easier, but it would protect the vast majority of the people, who has a legal right to want that their information would be protected [84].
Data and personal privacy is always a big concern when it comes to any type of technology which uses data, however, autonomous cars will have an "access" to people personal lives starting with car's location to environ them around it. With roads full of sponsored cars there will be no way that someplace and information about it will not be recorded, processed and stored. Due to that ensure safe storage for data is one of the most crucial tasks [84].

# 7 Conclusion and Future Work

## 7.1 Conclusion

In this work, we described and proposed probabilistic intention prediction algorithm, which can be used with frameworks (such as Markov decision processs (MDPs), Partially observable Markov decision processs (POMDPs), etc.), which requires to have a prediction model. The proposed algorithm is working based on an initial probabilistic model which is learned at initial stage form the demonstrations made before. With a learned initial probabilistic model, the proposed algorithm is able to predict the future motions, having the current position by conditioning the mean and variance values from predefined trajectories of each class to match the early observed data points. The proposed algorithm has some additional extensions: it is able to work in different environments: X and T intersection, with a different initial probabilistic model. As well, the algorithm was tested with a different type of testing trajectories: *previously prerecorded trajectory* - where we have a full trajectory and *real time trajectory* - where we are getting the position of the car real time while controlling it through visualization tool. Understanding that computational time and precision is one of the most important things while making a prediction, we proposed and described trajectory scaling, with which we achieved the more precision of prediction, but with less time running the algorithm. And eventually, using ProMPs we tried to predict how full trajectory looks like while only having early observations of the movement. Chapter 3 contains all theoretical framework of what working principles of the algorithm are.
Experiments were run with both: the real and the simulated data sets. In our experiments, the algorithm learns a set of motions respectively to different movement classes, from a number of demonstrations provided by us before. Having this prior knowledge and probabilistic model, the algorithm is able to make predictions about the intention of the car moving. When the car starts to drive, the probabilistic intention prediction algorithm uses observations to understand which movement class is chosen and which future direction is going to be.
Considering that security is a very important part of autonomous driving we made an overview of how it is possible to trick sensors to give false data to an algorithm which are making motion intention predictions. Since our algorithm is using only position data, only attacks on GPS are common (but for now these attacks are more experimental and made only on research purposes, only the matter of time when they will become real). However, to make predictions more accurate and while implementing suggestions in future work, it would be wise to consider cyber attacks which are described in Chapter 6.
Even though the probabilistic intention prediction algorithm is quite accurate while making predictions, it can always be improved. The section below described future works for improving the algorithm.

## 7.2 Future Work

While our proposed probabilistic intention prediction algorithm has quite precise results and some advantages in predicting intentions, there are the number of improvements which can be done to make the method even better.
**Improving Prediction Making Process:** One of the most important thing for future investigations is to answer the question of how to improve prediction making process by adding more information to the algorithm. Even though to consider position of the car alone can be enough for a simulated environment, for prediction making in the real world it can be too less information. Additionally having visual and vehicle dynamic information can help to improve intention prediction.
**Changing the Starting Point of the Car:** We made all our test from the same direction with a small inaccuracy for a starting point. We did not try to start moving from other direction on the map, so it would be nice to test algorithm when e.g. the starting point of the car is upper right wing in T intersection and available directions are straight and left.
**Prediction With Error:** Our proposed algorithm was tested in two maps environments: X and T. T intersection have only two directions, in our case, it was left and right. Predictions for future intentions were made quite accurate, but we did not check what is happening if due to some reasons an algorithm predicts wrongly and says that car is going to straight (when there is no straight). What is happening when the prediction is wrong? How to re-predict the direction, having wrong data? If the "car is unsure" about the correctness of prediction what should it do: continue to drive or stop? These questions need to be considered while improving probabilistic intention prediction algorithm in the future.
**Increase Level of Security:** When the proposed algorithm will be improved with additional steps to consider before making a prediction, e.g. some information from sensors, such as a camera or LiDAR it is important to consider attacks against sensors which can result with sensors giving the wrong information to algorithm which will cause the wrong

prediction and wrong decision making. This false information can end up with accidents and can cause a lot of problems for all road users and for traffic. To make sure that system is secure against attack is a very important task to make.
With this thesis, we did a literature review on the most common attacks on sensors and possible ways to avoid it. Since no security was implemented it is important to consider this task with future development of the algorithm.
**More Testing Environments:** We tested our algorithm only on T and X intersections. For future works, more city-like maps can be added to which combine solutions for T and X intersections together.
**Improving Computation Time:** Computation time for critical systems is crucial. 0.5s delay, which for a human can look not so important, can make a huge difference in machines. So any improvement considering time can make a big improvement in algorithm adaptation.

# Bibliography


[1] C. L. Stéphanie Lefèvre, Dizan Vasquez, "A survey on motion prediction and risk assessment for intelligent vehicles," *ROBOMECH Journal*, 2014.

[2] V. L. Thing and J. Wu, "Autonomous Vehicle Security: A Taxonomy of Attacks and Defences," 2016.

[3] I. G. S. J. Z. B. C. Nicolas Papernot, Patrick McDaniel and A. Swami, "Practical Black-Box Attacks against Machine Learning," 2017.

[4] E. F. B. L. A. R. C. X. A. P. T. K. Kevin Eykholt, Ivan Evtimov and D. Song, "Robust Physical-World Attacks on Deep Learning Visual Classification," 2018.

[5] F. STAFF, "Infographic: Data and the Connected Car – Version 1.0."

[6] S. R. Kaiming He, Xiangyu Zhang and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," 2015.

[7] T. Keeney, "Mobility-As-A-Servive: Why Self-Driving Cars Could Change Everything," 2017.

[8] H. R. S. Zhou Wang, Alan Conrad Bovik and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," p. 600–612, 2004.

[9] M. P. D. H. B. A. D. V. B. S. Zhikun Wang, Katharina Mülling and J. Peters, "Probabilistic movement modeling for intention inference in human-robot interaction," p. 841–858, 2013.

[10] C. F. Angelika Zube, Jonas Hofmann, "Model Predictive Contact Control for Human-Robot Interaction," p. 279–285, 2016.

[11] D. B. Sylvain Calinon and D. G. Caldwell1, "A Task-Parameterized Probabilistic Model with Minimal Intervention Control," p. 3339–3344, 2014.

[12] G. B. Ismail Dagli, Michael Brost, "Action recognition and prediction for driver assistance systems using dynamic belief networks," p. 179–194, 2002.

[13] K. T. Shigeki Tezuka, Hitoshi Soma, "A study of driver behavior inference model at time of lane change using bayesian networks," p. 2308–2313, 2006.

[14] A. P. Nuria Oliver, "Driver behavior recognition and prediction in a smartcar,," p. 280–290, 2000.

[15] A. P. Nuria Oliver, "Graphical models for driver behavior recognition in a smartcar," p. 7–12, 2000.

[16] F. Meier and S. Schaal, "A Probabilistic Representation for Dynamic Movement Primitives," 2016.

[17]

[18] ROS, "About ROS."

[19] TuD, "aDDa for Students."

[20] A. P. Andrew Liu, "Towards real-time recognition of driver intentions," p. 236–241, 1997.

[21] A. P. Andrew Liu, "Modeling and prediction of human behavior," p. 229–242, 1999.

[22] D. D. Salvucci, "Inferring driver intent: A case study in lane-change detection," p. 2228–2231, 2004.

[23] C. S. Yi Hou, Praveen Edara, "A genetic fuzzy system for modeling mandatory lane changing," 2012.

[24] D. D. S. Hiren M. Mandalia, "Using support vector machines for lane change detection," 2005.

[25] A. R. Nachiket Deo and M. M. Trived, “How Would Surround Vehicles Move? A Unified Framework for Maneuver Classification and Motion Prediction,” 2018.

[26] J. S. Mattias Brannstrom, Erik Coelingh, “Model-Based Threat Assessment for Avoiding Arbitrary Vehicle Collisions,” p. 658–669, 2010.

[27] K. K. Jrg Hillenbrand, Andreas M. Spieker, “A multilevel collision mitigation approach: situation assessment, decision making, and performance tradeoffs,” p. 528–540, 2006.

[28] A. J. A. L. A. Aris Polychronopoulos, Manolis Tsogas, “Sensor fusion for predicting vehicles’ path for collision avoidance systems,” p. 549–562, 2007.

[29] F. N. Samer Ammoun, “Real time trajectory prediction for collision risk estimation between vehicles ,” p. 417–422, 2009.

[30] M. S. Nico Kaempchen, Kilian Weiß, “IMM object tracking for high dynamic driving maneuvers,” p. 825–830, 2004.

[31] J. B. Thomas Batz, Kym Watson, “Recognition of dangerous situations within a cooperative group of vehicles,” p. 907–912, 2009.

[32] K. P. Murphy, “Dynamic Bayesian networks: representation, inference and learning,” 2009.

[33] S. B. Adrian Broadhurst and T. Kanade, “Monte Carlo road safety reasoning,” p. 319–324, 2005.

[34] A. M. Matthias Althoff, “Comparison of Markov chain abstraction and Monte Carlo simulation for the safety assessment of autonomous cars.,” p. 1237–1247, 2011.

[35] D. from Wordreference, “Definition of maneuver.”

[36] S. B. Tobias Gindele and R. Dillmann, “A probabilistic model for estimating driver behaviors and vehicle trajectories in traffic environments ,” p. 1625–1631, 2010.

[37] D. R. Ismail Dagli, “Motivation-based approach to behavior prediction,” p. 227–233, 2002.

[38] L. H. S. J. P. H. Georges S. Aoude, Vishnu R. Desaraju, “Driver behavior classification at intersections and validation on large naturalistic dataset,” p. 724–736, 2012.

[39] C. Laugier *et al.*, “Probabilistic analysis of dynamic scenes and collision risks assessment to improve driving safety,” p. 4–19, 2011.

[40] A. W. G. B. J. Schlechtriemen, F. Wirthmueller and K.-D. Kuhnert, “When will it change the lane? A probabilistic regression approach for rarely occurring events,” p. 1373–1379.

[41] G. B. J. Schlechtriemen, A. Wedel and K.-D. Kuhnert, “A probabilistic long term prediction approach for highway scenarios,” p. 732–738, 2014.

[42] V. C. Y. W. Adam Houenou, Philippe Bonnifait, “Vehicle Trajectory Prediction based on Motion Model and Maneuver Recognition,” p. 4363–4369, 2013.

[43] K. K. H. L. D. S. L. J. P. H. Georges S. Aoude, Brandon D. Luders, “Threat assessment design for driver assistance system at intersections,” p. 1855–1862, 2010.

[44] M. M. T. Brendan Tran Morris, “Trajectory learning for activity understanding: Unsupervised, multilevel, and long-term adaptive approach,” p. 2287–2301, 2011.

[45] K. D. Holger Berndt, Jorg Emmert, “Continuous driver intention recognition with hidden Markov models,” p. 1189–1194, 2008.

[46] M. M. T. Aida Khosroshahi, Eshed Ohn Bar, “Surround vehicles trajectory analysis with recurrent neural networks,” p. 2267–2272, 2016.

[47] J. A. Matthias Schreier, Volker Willert, “Bayesian, maneuver-based, longterm trajectory prediction and criticality assessment for driver assistance systems,” p. 334–341, 2014.

[48] J. F. Quan Tran, “Online maneuver recognition and multimodal trajectory prediction for intersection assistance using non-parametric regression,” p. 918–923, 2014.

[49] C. W. H. R. E. Kfer, C. Hermes and F. Kummert, “Recognition of situation classes at road intersections,” pp. 3960–3965, 2010.

[50] C. F. P. G. T. D. W. A. Lawitzky, D. Althoff and M. Buss, “Interactive scene prediction for automotive applications,” pp. 1028–1033, 2013.

[51] K. S. C. Hermes, C. Wohler and F. Kummert, “Long-term vehicle motion prediction,” p. 652–657, 2009.

[52] D. Vasquez and T. Fraichard, “Motion prediction for moving objects: a statistical approach,” p. 3931–3936, 2004.

[53] F. D.-V. J. M. Joseph and N. Roy, “A Bayesian nonparametric approach to modeling mobility patterns,” p. 1587–1593, 2010.

[54] U. K. J. Wiest, M. Hffken and K. Dietmayer, “Probabilistic trajectory prediction with Gaussian mixture models,” p. 141–146, 2012.

[55] N. R. J. P. H. Georges S. Aoude, Joshua Joseph, “Mobile agent trajectory prediction using Bayesian nonparametric reachability trees,” p. 1587–1593, 2011.

[56] OpenCV, “About OpenCV.”

[57] OpenCV, “Contours in OpenCV.”

[58] Wikipedia, “Ramer–Douglas–Peucker algorithm.”

[59] H. R. S. Zhou Wang, Alan Conrad Bovik and E. P. Simoncelli, “Image Quality Assessment: From Error Visibility to Structural Similarity,” p. 600–612, 2004.

[60] A. Rosebrock, “How-To: Python Compare Two Images.”

[61] M. J. Kochenderfer, “Decision Making Under Uncertainty: Theory and Application,” p. 11–57, 2015.

[62] A. R. C. Leslie Pack Kaelbling, Michael L. Littman, “Planning and acting in partially observable stochastic domains,” p. 99–134, 1998.

[63] S. Scholar, “Toy Problem.”

[64] M. E. R. L. O. K. Guilherme Maeda, Gerhard Neumann and J. Peters, “Probabilistic Movement Primitives for Coordination of Multiple Human-Robot Collaborative Tasks,” p. 1–20, 2016.

[65] R. L. H. B. A. J. P. Marco Ewerton, Gerhard Neumann and G. Maeda, “Learning multiple collaborative tasks with a mixture of interaction primitives,” p. 1535–1542, 2015.

[66] ROS, “ROS joy package summary.”

[67] I. M. Institute, “Self Driving Car Security.”

[68] D. M. West, “Moving forward: Self-driving vehicles in China, Europe, Japan, Korea, and the United States.”

[69] N. A. S. G. D. K. Swarun Kumar, Lixin Shi and D. Rus, “CarSpeak: A Content-Centric Network for Autonomous Driving,” 2012.

[70] F. R. S. P. Karl Koscher, Alexei Czeskis and T. Kohno, “Experimental Security Analysis of a Modern Automobile,” 2010.

[71] D. Lagutin, “Packet Level Authentication Overview,” 2010.

[72] K.-T. Cho and K. G. Shin, “Fingerprinting electronic control units for vehicle intrusion detection,” 2016.

[73] J. A. B. T. E. H. Andrew J. Kerns, Daniel P. Shepard, “Unmanned aircraft capture and control via GPS spoofing,” p. 617–636, 2014.

[74] A. Couts, “Want to see this 80 million super yacht sink? With GPS spoofing, now you can!.”

[75] F. K. Jonathan Petit, Michael Feiri, “Remote Attacks on Automated Vehicles Sensors: Experiments on Camera and LiDAR,” 2015.

[76] M. T. R. Mostofa Kamal Nasir, A.K.M. Kamrul Islam and M. K. Sohel, “Taxonomy of Security in Vehicular Ad-Hoc Network,” 2013.

[77] M.-J. Kang and J.-W. Kang, “Intrusion detection system using deep neural network for in-vehicle network security.”

[78] F. K. Jonathan Petit, Michael Feiri, “Revisiting Attacker Model for Smart Vehicles,” p. 1–5, 2014.

[79] H. M. Xuesong Mao, Daisuke Inoue and M. Kagami, “Demonstration of In-Car Doppler Laser Radar for Range and Speed Measurement,” p. 599–607, 3013.

[80] E. Ackerman, “Slight Street Sign Modifications Can Completely Fool Machine Learning Algorithms.”

[81] A. LaFrance, “How Self-Driving Cars Will Threaten Privacy.”

[82] A. Marshall, “Congress Units (GASP) to Spread Self-Driving Cars Across America.”

[83] A. Kharpal, “Apple vs FBI: All you need to know.”

[84] B. Cresitello-Dittmar, “Privacy Concerns of Self Driving Cars.”