Autonomous vehicle localization using state estimation techniques

Master Thesis

April 22, 2020

Author: Dunia Prieto Francia
Supervisor: Vicenç Puig Cayuela
Call: April 23, 2020
Abstract

Recently, industrialized countries carry out a technological race towards autonomous driving. The current commercial driver assistant systems are limited to aiding the driver while the driver is responsible for overall control of the vehicle or some autonomous driving functions under certain driving conditions. There are still many challenges to be solved to allow autonomous vehicles to emerge into the market. One of the key challenges for the development of fully autonomous vehicles is localizing the vehicle in known, unknown, or uncertain environments.

Most of the existing works in the field of mobile robotics rely on a non-linear kinematic motion model under the assumption that the velocity is low. However, in autonomous vehicle applications where the velocity is higher, a dynamic model is required. On the other hand, conventional solutions are based on Extended Kalman filter that performs a linearization around the estimated trajectory.

Within this context, this master thesis has considered a non-linear kinematic/dynamic motion model. Furthermore, the use of an LPV Kalman filter for estimating the vehicle dynamic state has been explored. Due to the philosophy underlying the design procedure, computational complexity is reduced and strong non-linearities are easier to handle. Kinematic states are obtained by means of numerical integration.

The proposed approach has been implemented and integrated in an SLAM Toolbox for Matlab. Simulations results validate the developed localization technique. However, the results still show space for improvement. In particular, a better integration between the kinematic and dynamic parts should be developed. This can be achieved by designing an observer that integrates both the dynamic and kinematic parts. Another extension that is planned is the inclusion of landmarks in the LPV Kalman observer and the map building capability.

Reaching autonomous driving it just a matter of time!!!
Autonomous vehicle localization using state estimation techniques

Contents

1 Introduction 1
  1.1 Context and motivation ................................................. 1
  1.2 Objectives ...................................................................... 2
  1.3 Structure of the document .............................................. 2

2 State-of-the-art of localization techniques 3
  2.1 Basic terminology in probabilistic localization ....................... 3
  2.2 Background in probabilistic localization techniques [2] .......... 4
    2.2.1 Gaussian filters ..................................................... 6
      2.2.1.1 The Kalman Filter (KF) algorithm ......................... 6
      2.2.1.2 The Extended Kalman Filter (EKF) algorithm ........... 7
  2.3 EKF localization with odometry motion model [1] .......... 8
    2.3.1 Modeling the vehicle ................................................ 9
    2.3.2 Estimating the pose ................................................ 10

3 Statement of the addressed vehicle localization problem 12
  3.1 Autonomous vehicle case study ........................................ 12
  3.2 Kinematic and dynamic vehicle models ([8], section 2) .......... 13
    3.2.1 Kinematic model .................................................... 13
    3.2.2 Dynamic model ..................................................... 14
      3.2.2.1 Calculation of lateral forces using Pacejka’89 tyre model ... 15
  3.3 Dynamic LPV modelling .................................................. 17
  3.4 Odometry-based localization using LPV Kalman filter .......... 18
  3.5 Optimal solution to Matrix Ricatti Equation for KF implementation ... 19

4 Solution based on odometry and LPV Kalman filter 21
  4.1 LPV Kalman filter design based on polytopic approximation ...... 21
  4.2 Calculation of polytopic observer gains using LMIs .............. 21
  4.3 Dynamic LPV Kalman filter design ..................................... 22
  4.4 Pose integration .......................................................... 22
  4.5 Comparison between LPV and Extended Kalman filter .......... 22

5 Integration of LPV Kalman Filter localization algorithms in SLAM Toolbox 23
  5.1 Introduction to SLAM Toolbox in Matlab ............................ 23
  5.2 Matlab implementation of LPV Kalman Filter and dynamic vehicle model .. 23
  5.3 Integration of vehicle localization algorithm in Graph-SLAM Toolbox .......... 24

6 Simulation results .............................................................. 27

7 Conclusions and future work ............................................... 31

Acknowledgments .............................................................. 32

Bibliography ....................................................................... 33
## List of Figures

1. The dynamic Bayes network that characterizes the evolution of controls, states, and measurements. ........................................ 5
2. The general algorithm for Bayes filtering. ........................................ 5
3. The Kalman filter algorithm for linear Gaussian state transitions and measurements. .................................................. 7
4. The extended Kalman filter algorithm. ........................................... 8
6. Electric Tazzari Zero vehicle used as case study ................................. 12
7. Control architecture of the Elektra vehicle ......................................... 13
8. Two-wheels bicycle model. \{W\} frame represents the global inertial frame and \{B\} is the body frame located in the centre of gravity of the vehicle. .............. 14
9. Tyre operating at a slip angle. ....................................................... 16
10. Lateral force versus slip angle. ...................................................... 16
11. Proposed circuit for simulation and the result of solving the localization problem .......................... 27
12. Control actions used as system inputs: a) longitudinal rear force, b) steering angle. ................. 27
13. Results of applying the dynamic LPV Kalman filter for solving the localization problem: a) linear velocity, b) angular velocity. .......................... 29
14. Results of applying the dynamic LPV Kalman filter for solving the localization problem: slip angle. ........................................ 29
15. Position errors referred to global fixed coordinate system \{W\}. ................. 30
16. Vehicle orientation error in the z-axis. .......................................... 30
List of Tables
1   Comparison between Kalman filter and EKF algorithm . . . . . . . . . . . . . . . 8
2   Kinematic and dynamic model parameters . . . . . . . . . . . . . . . . . . . . . . 14
3   Pacejka Magic Formula parameters . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
4   Matlab files implementing autonomous vehicle localization techniques . . . . . 24
1 Introduction

1.1 Context and motivation

In the past three decades, there has been great interest and progress in the field of intelligent vehicles for both researchers and industries. The continuous development of sensing and computation technologies has led to the development of various driver assistant systems. Some of these driver assistant systems, such as lane keeping, lane changing, adaptive cruise control, and highway driver assistant systems have already been installed on vehicles.

There are six different levels of driver assistant systems ranging from fully human operated to fully automated [5]. The current commercial driver assistant systems are limited to aiding the driver while the driver is responsible for overall control of the vehicle or some autonomous driving functions under certain driving conditions. There are still many challenges to be solved to allow autonomous vehicles to emerge into the market.

The architecture through which autonomous driving is achieved can be described by five functional systems, including localization, perception, planning, control, and system management [4]. The localization system identifies the location of the vehicle on a global coordinate system while the perception system evaluates the driving environment around the vehicle and identifies elements such as other road users, traffic signals, and obstacles. The planning system uses the inputs from the perception and localization systems to determine the travel paths and driving actions such as lane changes, accelerating, and braking. The control system transforms the behaviour and actions indicated by the planning system to the vehicle manipulation commands (e.g., steering, accelerating, and braking). The system management oversees the operation of all the systems and provides the human–machine interface. These functional systems must cooperate to answer the three important questions of "Where is the car?", "What is around the car?" and "What does the car need to do next?" to achieve fully autonomous operation.

The perception, planning, and control functional systems require accurate knowledge of the vehicle’s location to carry out the correct driving decisions and actions. For example, an error of few decimeters can cause the vehicle to localize itself on the wrong side of the road or can cause accidents to the vulnerable users such as pedestrians and cyclists. Robustness is also required as the vehicle needs to localize itself in uncertain driving conditions (e.g., in cases where road markings are absent or obscured) and in harsh environmental conditions (e.g., darkness and snow). Therefore, autonomous vehicles require robust localization systems with centimeter level of accuracy.

Most of the existing works in the field of mobile robotics rely on a non-linear kinematic model under the assumption that the velocity is low. However, in autonomous vehicle applications where the velocity is higher, a dynamic model is required. On the other hand, the use of Extended Kalman Filter (EKF), which is based on a linearization around the estimated trajectory, is predominant. Hence, this master thesis will consider the use of an odometry-based Linear Parameter Varying (LPV) Kalman filter using a non-linear kinematic/dynamic model for estimating the position of an autonomous vehicle. LPV Kalman filters are capable of dealing with the non-linearities by embedding them inside model parameters that depend on some variables, called scheduling parameters, that vary in a known bounded interval [12]. Moreover, the design of the LPV Kalman filter is based on the use of Linear Matrix Inequalities (LMIs) and Lyapunov theory, which provides guarantee of convergence (stability) of the estimations.
1.2 Objectives
Taking into account the motivations mentioned above, the following objectives will be addressed along this master thesis:

- Representation of the kinematic/dynamic model of the autonomous vehicle in LPV form using the non-linear embedding approach.
- Design of an LPV Kalman filter using the polytopic representation of the LPV model of the vehicle and the LMI framework.
- Application of the LPV Kalman filter to the autonomous vehicle localization using odometry sensors.
- Integration of the proposed vehicle localization technique in a Graph-based SLAM toolbox in Matlab.
- Validation of the solution using the simulation environment of the SLAM toolbox conveniently adapted for this purpose.

1.3 Structure of the document
The remainder of this master thesis is structured as follows. Chapter 2 describes current state of autonomous vehicle localization techniques. The odometry-based vehicle localization problem using a kinematic/dynamic model is formally described in Chapter 3, while Chapter 4 develops the proposed solution based on the use of an LPV Kalman filter. Chapter 5 explains the details of the integration of the non-linear kinematic/dynamic vehicle motion model and the LPV Kalman filter within a Graph-based SLAM toolbox in Matlab. Some simulation results that illustrate the performance of the proposed solution are provided in Chapter 6. Finally, Chapter 7 draws the conclusions and poses future extensions of the proposed method in the sense of designing an observer that integrates both the dynamic and kinematic parts as well as including the use of landmarks.
2 State-of-the-art of localization techniques

Localization is the problem of estimating the coordinates of an object, in the context of this work a vehicle (or robot), relative to an external reference frame.

Not every localization problem is equally hard. Localization problems can be classified according to different criteria pertaining to the nature of the environment and the initial knowledge that a vehicle may possess ([2], chapter 7). Along this master thesis, we will consider vehicle position tracking in static environments, which assumes that the initial robot pose is known and the robot localization can be achieved by accommodating the noise in robot motion. All other objects in the environment remain at the same location forever.

A variety of sensors and instrumentation are utilized in intelligent vehicles for localization [5]. These sensors can be classified into two classes: proprioceptive and exteroceptive.

- The proprioceptive sensors internally measure and collect data, such as velocity and steering angle of the vehicle. Most of the proprioceptive sensors used for localization and navigation system purposes are onboard motion sensors (such as encoders and Inertial Measurement Units (IMUs)). These sensors can only allow self-localization and cannot be used to find the location of other objects.

- The exteroceptive sensors aim to measure information from the vehicle environment. Most of the exteroceptive sensors are used for localization relative to the immediate surrounding of the vehicle, although via feature extraction some of the relative localization sensors can also be used for absolute localization. Some examples are range and vision sensors.

However, sensors are noisy, and there are usually many variables that cannot be sensed directly. This is because of the inaccuracy and incompleteness of these sensors that localization poses difficult challenges. To enhance the performance of localization algorithms and achieve reliability and accuracy, multiple sensors are used and fused together.

The remainder of this chapter is organized as follows. Key concepts and mathematical tools for estimating state from sensor data are introduced in Sections 2.1 and 2.2, respectively. Finally, a particular implementation of the extended Kalman filter for odometry motion model is described in Section 2.3.

2.1 Basic terminology in probabilistic localization

Environments are characterized by state. Throughout this work, state will be denoted \( x \); although the specific variables included in \( x \) will depend on the context. The state at discrete time \( k \) will be denoted \( x_k \). Most commonly used state variables are:

- The robot pose, which comprises its location and orientation relative to a global coordinate frame. Rigid mobile robots possess six such state variables, three for their Cartesian coordinates (\( x, y, \) and \( z \)), and three for their angular orientation (pitch, roll, and yaw). For rigid mobile robots confined to planar environments, the pose is usually given by three variables, its two location coordinates in the plane (\( x \) and \( y \)) and its heading direction (yaw).

- The robot velocity is commonly referred to as dynamic state. A rigid robot moving through
space is characterized by up to six velocity variables, one for each pose variables.

- The location and features of surrounding objects in the environment are also state variables. In some problems, objects will assume the form of landmarks, which are distinct, stationary features of the environment that can be recognized reliably. The state of the map is denoted as \( m \).

A state \( x_k \) will be called complete if it is the best predictor of the future.

There are two fundamental types of interactions between a robot and its environment:

- **Environment sensor measurements.** Perception is the process by which the robot uses its sensors to obtain information about the state of its environment. The result of such a perceptual interaction will be called a measurement or observation. The measurement data at time \( k \) will be denoted \( z_k \).

- **Control actions** change the state of the world. Examples of control actions include robot motion and the manipulation of objects. For consistency, we will assume that the robot always executes a control action, even if it chooses not to move any of its motors. Control data will be denoted \( u_k \). The variable \( u_k \) will always correspond to the change of state in the time interval \( ((k - 1)T_s, kT_s] \) where \( T_s \) is the sampling time.

For mobile robots operating in the plane, either velocity or odometry motion models are employed \([2]\). In practice, odometry models tend to be more accurate than velocity models, for the simple reason that most commercial robots do not execute velocity commands with the level of accuracy that can be obtained by measuring the revolution of the robot’s wheels. However, odometry is only available after executing a motion command. Hence, it cannot be used for motion planning.

In practice, the robot continuously executes controls and measurements are made concurrently. Environment perception provides information about the environment’s state, hence it tends to increase the robot’s knowledge. Motion, on the other hand, tends to induce a loss of knowledge due to the inherent noise.

### 2.2 Background in probabilistic localization techniques \([2]\)

Probabilistic robotics is a relatively new approach to robotics that pays tribute to the uncertainty in robot perception and action. The evolution of state and measurements is governed by probabilistic laws: the state transition distribution, and the measurement distribution.

The probability \( p(x_k|x_{k-1}, u_k) \) is the state transition probability. It specifies how environmental state evolves over time as a function of robot controls \( u_k \). Robot environments are stochastic, which is reflected by the fact that \( p(x_k|x_{k-1}, u_k) \) is a probability distribution, not a deterministic function. The probability \( p(z_k|x_k) \) is called the measurement probability. It is appropriate to think of measurements as noisy projections of the state.

By considering \( p(x_k|x_{k-1}, u_k) \), we defined robot motion in a vacuum. In particular, this model describes robot motion in the absence of any knowledge about the nature of the environment. In many cases, we are also given a map \( m \), which may contain information pertaining to the places that a robot may or may not be able to navigate.
The state transition probability and the measurement probability together describe the dynamical stochastic system of the robot and its environment. Figure 1 illustrates the evolution of states and measurements, defined through those probabilities. The state at time \( k \) is stochastically dependent on the state at time \( k - 1 \) and the control \( u_k \). The measurement \( z_k \) depends stochastically on the state at time \( k \). Such a temporal generative model is also known as *dynamic Bayes network* (DBN).

![Figure 1: The dynamic Bayes network that characterizes the evolution of controls, states, and measurements.](image)

Another key concept in probabilistic robotics is that of a *belief*. A belief reflects the robot’s internal knowledge about the state of the environment. Belief distributions are posterior probabilities over state variables conditioned on the available data. We will denote belief over a state variable \( x_k \) by \( \text{bel}(x_k) \), which is an abbreviation for the posterior \( \text{bel}(x_k) = p(x_k | z_{1:k}, u_{1:k}) \). Occasionally, it will prove useful to calculate a posterior before incorporating the measurement \( z_k \), just after executing the control \( u_k \). Such a posterior will be denoted as follows: \( \text{bel}(x_k) = p(x_k | z_{1:k-1}, u_{1:k}) \). This probability distribution is often referred to as *prediction* in the context of probabilistic filtering. Calculating \( \text{bel}(x_k) \) from \( \text{bel}(x_k) \) is called *correction* or the *measurement update*.

The most general algorithm for calculating beliefs is given by the *Bayes filter* algorithm. Figure 2 depicts the basic Bayes filter in pseudo-algorithmic form. The Bayes filter is recursive, that is, the belief \( \text{bel}(x_k) \) at time \( k \) is calculated from the belief \( \text{bel}(x_{k-1}) \) at time \( k - 1 \). Its input is the belief \( \text{bel} \) at time \( k - 1 \), along with the most recent control \( u_k \) and the most recent measurement \( z_k \). Its output is the belief \( \text{bel}(x_k) \) at time \( k \). Figure 2 only depicts a single iteration of the Bayes Filter algorithm: the *update rule*. The initial belief, \( \text{bel}(x_0) \), reflects the initial knowledge of the robot’s pose. The Bayes filter makes a *Markov assumption* according to which the state is a complete summary of the past. This assumption implies the belief is sufficient to represent the past history of the robot.

```
1: Algorithm Bayes_filter(bel(x_{t-1}), u_t, z_t):
2:   for all x_t do
3:     bel(x_t) = \int p(x_t | u_t, x_{t-1}) \ bel(x_{t-1}) \ dx_{t-1}
4:     bel(x_t) = \eta p(z_t | x_t) \ bel(x_t)
5:   endfor
6: return bel(x_t)
```

Figure 2: The general algorithm for Bayes filtering.
2.2.1 Gaussian filters

Historically, Gaussian filters constitute the earliest tractable implementations of the Bayes filter for continuous spaces. Gaussian techniques all share the basic idea that beliefs are represented by multivariate normal distributions characterized by density functions of the following form:

\[ p(x) = \text{det}(2\pi \Sigma)^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right\} \]  

(1)

The density over the variable \( x \) is characterized by two sets of parameters: The mean \( \mu \) and the covariance \( \Sigma \). The mean \( \mu \) is a vector that possesses the same dimensionality as the state \( x \). The covariance is a quadratic matrix that is symmetric and positive-semidefinite. Its dimension is the dimensionality of the state \( x \) squared. The parameterization of a Gaussian by its mean and covariance is called the moments parameterization. This is because the mean and covariance are the first and second moments of a probability distribution; all other moments are zero for normal distributions. Most importantly, Gaussians are unimodal; they possess a single maximum. Such a posterior is characteristic of position tracking problems, in which the posterior is focused around the true state with a small margin of uncertainty.

The Kalman filter, which implements the Bayes filter using the moments parameterization for a restricted class of problems with linear dynamics and measurement functions is introduced in section 2.2.1.1. Next, its extension to non-linear problems is described in section 2.2.1.2.

2.2.1.1 The Kalman Filter (KF) algorithm

The Kalman filter is a technique for filtering and prediction in linear Gaussian systems, which will be defined in a moment. Beliefs are represented by the moments parameterization: At time \( k \), the belief is represented by the the mean \( \mu_k \) and the covariance \( \Sigma_k \). Posteriors are Gaussian if the following three properties hold, in addition to the Markov assumptions of the Bayes filter:

1. Kalman filters assume linear system dynamics. This is expressed by the following equation:

\[ x_{k+1} = A_k x_k + B_k u_k + \varepsilon_k \]  

(2)

\( A_k \) and \( B_k \) are matrices. \( A_k \) is a square matrix of size \( n_x \times n_x \), where \( n_x \) is the dimension of the state vector \( x_k \). \( B_k \) is of size \( n_x \times n_u \), with \( n_u \) being the dimension of the control vector \( u_k \). The random variable \( \varepsilon_k \) is a Gaussian random vector that models the uncertainty introduced by the state transition. It is of the same dimension as the state vector. Its mean is zero, and its covariance will be denoted \( R_k \).

2. The measurement probability \( p(z_k|x_k) \) must also be linear in its arguments, with added Gaussian noise:

\[ z_k = C_k x_k + \delta_k \]  

(3)

Here \( C_k \) is a matrix of size \( n_z \times n_x \), where \( n_z \) is the dimension of the measurement vector \( z_k \). The vector \( \delta_k \) describes the measurement noise. The distribution of \( \delta_k \) is a multivariate Gaussian with zero mean and covariance \( Q_k \).

3. Finally, the initial belief \( \text{bel}(x_0) \) must be normally distributed. We will denote the mean of this belief by \( \mu_0 \) and the covariance by \( \Sigma_0 \).
The Kalman filter algorithm is depicted in Figure 3. Kalman filters represent the belief $\text{bel}(x_k)$ at time $k$ by the mean $\mu_k$ and the covariance $\Sigma_k$. The input of the Kalman filter is the belief at time $k-1$, represented by $\mu_{k-1}$ and $\Sigma_{k-1}$. To update these parameters, Kalman filters require the control $u_k$ and the measurement $z_k$. The output is the belief at time $k$, represented by $\mu_k$ and $\Sigma_k$.

The variable $K_k$, computed in line 4 of Figure 3 is called Kalman gain. It specifies the degree to which the measurement is incorporated into the new state estimate. The key concept here is the innovation, which is the difference between the actual measurement $z_k$ and the expected measurement $C_k \mu_k$ in line 5.

1: Algorithm Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2: $\mu_t = A_t \mu_{t-1} + B_t u_t$
3: $\Sigma_t = A_t \Sigma_{t-1} A_t^T + R_t$
4: $K_t = \Sigma_t C_t (C_t \Sigma_t C_t^T + Q_t)^{-1}$
5: $\mu_t = \mu_t + K_t (z_t - C_t \mu_t)$
6: $\Sigma_t = (I - K_t C_t) \Sigma_t$
7: return $\mu_t, \Sigma_t$

Figure 3: The Kalman filter algorithm for linear Gaussian state transitions and measurements.

### 2.2.1.2 The Extended Kalman Filter (EKF) algorithm

The assumptions that observations are linear functions of the state and that the next state is a linear function of the previous state are crucial for the correctness of the Kalman filter. Also, the fact that any linear transformation of a Gaussian random variable results in another Gaussian random variable plays an important role in the derivation of the Kalman filter algorithm. Unfortunately, state transitions and measurements are rarely linear in practice. This observation, along with the assumption of unimodal beliefs, renders plain Kalman filters inapplicable to all but the most trivial localization problems.

The EKF relaxes the linearity assumption. Here the assumption is that the state transition probability and the measurement probabilities are governed by non-linear functions $g$ and $h$, respectively:

$$
x_{k+1} = g(x_k, u_k) + \varepsilon_k \quad (4)
$$

$$
z_k = h(x_k) + \delta_k \quad (5)
$$

This model strictly generalizes the linear Gaussian model underlying Kalman filters. Even so, with arbitrary functions $g$ and $h$, the belief is no longer a Gaussian, and the Bayes filter does not possess a closed-form solution. To overcome this shortcoming, the EKF calculates a Gaussian approximation to the the true belief.

The key idea underlying the EKF approximation is called linearization. EKFs utilize a method called (first order) Taylor expansion. Taylor expansion constructs a linear approximation to a function $g$ from $g$’s value and slope. The slope is given by the partial derivative.
Notice that \( G_k \) is a matrix of size \( n_x \times n_x \), with \( n_x \) denoting the dimension of the state. This matrix is often called the Jacobian.

Since for Gaussians, the most likely state is the mean of the posterior \( \mu_k \), \( g \) is approximated by its value at \( \mu_k \) (and at \( u_k \)), and the linear extrapolation is achieved by a term proportional to the gradient of \( g \) at \( \mu_k \) and \( u_k \):

\[
g(u_k, x_k) \approx g(\mu_k, u_k) + G_k(\mu_k - x_k)
\]  

Projecting the Gaussian through this linear approximation results in a Gaussian density. The exact same linearization for the measurement function \( h \) is implemented by EKFs.

Figure 4 states the EKF algorithm. In many ways, this algorithm is similar to the Kalman filter algorithm stated in Figure 3. The most important differences are summarized in Table 1:

<table>
<thead>
<tr>
<th>Kalman filter</th>
<th>EKF</th>
</tr>
</thead>
<tbody>
<tr>
<td>state prediction (line 2)</td>
<td>( A_k \mu_k + B_k u_k )</td>
</tr>
<tr>
<td>measurement prediction (line 5)</td>
<td>( C_k \mu_k )</td>
</tr>
</tbody>
</table>

Table 1: Comparison between Kalman filter and EKF algorithm.

That is, the linear predictions in Kalman filters are replaced by their non-linear generalizations in EKFs. Moreover, EKFs use Jacobians \( G_k \) and \( H_k \) instead of the corresponding linear system matrices \( A_k, B_k, \) and \( C_k \) in Kalman filters. The Jacobian \( G_k \) corresponds to the matrices \( A_k \) and \( B_k \), and the Jacobian \( H_k \) corresponds to \( C_k \).

2.3 EKF localization with odometry motion model [1]

The extended Kalman filter localization algorithm, or EKF localization, is a special case of Markov localization. EKF localization represents beliefs \( \text{bel}(x_k) \) by their first and second moment, the mean \( \mu_k \) and the covariance \( \Sigma_k \).

The basic EKF algorithm was stated in Figure 4 in a rather abstract way. We have silently assumed the availability of an appropriate motion and measurement model, and have left unspecified a number of key variables in the EKF update. We now discuss a particular implementation.
of the EKF for empty feature-based maps. The adopted odometry motion model is defined in subsection 2.3.1, while the pose update procedure is described in subsection 2.3.2. Since the vehicle moves in a vacuum, no measurement model is needed.

2.3.1 Modeling the vehicle

As stated in Section 2.1, for rigid mobile robots operating in planar environments, the kinematic state or pose is usually given by three variables, its two-dimensional planar coordinates relative to an external coordinate frame, along with its angular orientation. Denoting the former as $x$ and $y$, and the latter by $\theta$, the pose of the robot is described by the following vector:

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$  \hspace{1cm} (8)

The pose of a mobile robot operating in a plane is illustrated in Figure 5.

![Figure 5: The global reference frame and the robot local reference frame [3].](image)

The first step in estimating the robot’s pose is to write a function, $g(\cdot)$, that describes how the vehicle’s configuration changes from one time step to the next. The odometry motion model uses odometry measurements: $\delta = (\delta_d, \delta_\theta)^T$, where $\delta_d$ is the distance traveled and $\delta_\theta$ is the change in heading over the preceding interval per each time step $k$ (or $kT_s$ in time).

We make a simplifying assumption that motion over each time step (sampling time) is small so the order of applying the displacements is not significant. We choose to move forward in the vehicle $x$-direction by $\delta_d$, and then rotate by $\delta_\theta$ giving the new configuration, which can be
represented concisely as:

\[
x_{k+1} = \begin{pmatrix} x_k + \delta_d \cos(\theta_k) \\ y_k + \delta_d \sin(\theta_k) \\ \theta_k + \delta_\theta \end{pmatrix}
\]

which gives the new configuration in terms of the previous configuration and the odometry.

In practice odometry is not perfect and we model the error by imagining a random number generator that corrupts the output of a perfect odometer. The measured output of the real odometer is the perfect, but unknown, odometry \((\delta_d, \delta_\theta)\) plus the output of the random number generator \((\nu_d, \nu_\theta)\). Such random errors are referred to as sensor noise.

The robot’s configuration at the next time step, including the odometry error, is

\[
x_{k+1} = g(x_k, \delta_k, \nu_k) = \begin{pmatrix} x_k + (\delta_d + \nu_d) \cos(\theta(k)) \\ y_k + (\delta_d + \nu_d) \sin(\theta(k)) \\ \theta_k + \delta_\theta + \nu_\theta \end{pmatrix}
\]

In the absence of any information to the contrary we model the odometry noise as \(\nu = (\nu_d, \nu_\theta)^T \sim N(0, V)\), a zero-mean multivariate Gaussian process with variance

\[
V = \begin{pmatrix} \sigma_d^2 & 0 \\ 0 & \sigma_\theta^2 \end{pmatrix}
\]

This constant matrix, known as covariance matrix, is diagonal which means that the errors in distance and heading are independent. Choosing a value for \(V\) is not always easy but we can conduct experiments or make some reasonable engineering assumptions.

### 2.3.2 Estimating the pose

The problem we face is how to estimate our new pose given the previous pose and noisy odometry. We want the best estimate of where we are and how certain we are about that. The mathematical tool that we will use is the extended Kalman filter, because our model of the vehicle’s motion is non-linear.

For this problem the state vector is the vehicle’s configuration in Eq.(8) and the prediction equations

\[
x_{k+1} = g(x_k, u_k) = g(x_k, \delta_k)
\]

\[
\bar{P}_{k+1} = G_x P(k) G_x^T + G_v \dot{V} G_v^T
\]

describe how the state and covariance evolve with time. \(P\) is a \(3 \times 3\) covariance matrix representing uncertainty in the estimated vehicle configuration. \(\dot{V}\) is our estimate of the covariance of the odometry noise which in reality we do not know. \(G_x\) and \(G_v\) are Jacobian matrices. They are obtained by differentiating Eq.(10) and evaluating the result at \(\nu = 0\) giving

\[
G_x = \left. \frac{\partial g}{\partial x} \right|_{\nu=0} = \begin{pmatrix} 1 & 0 & -\delta_d \sin(\theta_k) \\ 0 & 1 & \delta_d \cos(\theta_k) \\ 0 & 0 & 1 \end{pmatrix}
\]
which are functions of the current state and odometry.
3 Statement of the addressed vehicle localization problem

3.1 Autonomous vehicle case study

The autonomous vehicle case study used in this master thesis is a Tazzari Zero vehicle\(^1\) (see Figure 6). This autonomous vehicle is being developed in the context of the Elektra project, a joint project between Polytechnic University of Catalonia (Universitat Politècnica de Catalunya, UPC) and Autonomous University of Barcelona (Universitat Autònoma de Barcelona, UAB), devoted to develop an electric autonomous vehicle\(^2\).

![Electric Tazzari Zero vehicle used as case study](image)

This vehicle is a non-holonomic platform that behaves as a normal road vehicle. It has been equipped with a set of sensors and actuators, as well as with an on-board computer and an electronic control unit (ECU) that manages all software modules and communications between them. The diagram of the control architecture is presented in Figure 7. The vehicle has on-board an inertial measurement unit (IMU), a GPS and stereo cameras to obtain information about the environment and current vehicle state. The crude information provided by these sensors is treated with several algorithms to provide more informative data that can be used to understand the environment and localize the vehicle. On the other hand, a set of actuators are available to control the motion (steering and driven electric motors) as well as operating lights and doors. The rest of modules in Figure 7 (perception, localization, planning and control) compose the software for performing the autonomous guidance task\(^9\).

All the required algorithms run inside a trunk computer (6-core i7 5930K, 32GB DDR4) running ROS on GNU/Linux (Ubuntu distribution). An NVIDIA GTX Titan X board is used to run GPU-based algorithms for perception-image analysis. The ECU, based on a Cortex-M4 MCU, runs a custom embedded software which communicates the embedded computer control actions to the different car actuators (steering, throttle, brake, lights, horn), and reads the values of the car state sensors (steering, throttle, brake, speed, doors, battery). The communication net is based on CAN bus protocol. The control cycle is currently set to 100 ms, which is sufficient for running all required algorithms\(^9\).

\(^1\)http://www.tazzari-zero.com/
\(^2\)http://adas.cvc.uab.es/elektra/
3.2 Kinematic and dynamic vehicle models ([8], section 2)

This part describes the models of the considered autonomous vehicle case study which will be used later for developing the vehicle localization strategies. A mobile object can be described by using equations that represent the dynamic and kinematic behaviours. Unlike common mobile robots, urban autonomous vehicles are systems with larger mass and operating at a higher velocity. This is the reason why the use of dynamic models becomes indispensable. On one hand, in dynamic models the sum of forces existing over the vehicle are taken into account for computing the vehicle acceleration. The motion is generated by applying forces over the driven wheels and mass. Inertial and tyre parameters are considered. On the other hand, kinematic model is based on the velocity vector movement in order to compute longitudinal and lateral velocities referenced to a global inertial frame. External forces are not considered in this case. Note that, for both models, the two-wheels bicycle mode has been considered, as the one depicted in Figure 8. The two-wheels model employed does not consider roll, pitch and z motion, only yaw, x and y movements.

In this work, both models are presented and used in a decoupled way. Table 2 presents the characteristic vehicle parameters used in the case study vehicle considered in this thesis.

3.2.1 Kinematic model

Kinematic based model is widely used due to its low parameter dependency. It assumes null skidding and considers lateral force to be so small that can be neglected. Basically, it is a geometric way to compute vehicle position and orientation considering linear and angular velocities. The kinematic equations are
Figure 8: Two-wheels bicycle model. \( \{W\} \) frame represents the global inertial frame and \( \{B\} \) is the body frame located in the centre of gravity of the vehicle.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>Distance from CoG to front axle</td>
<td>0.758 m</td>
</tr>
<tr>
<td>( b )</td>
<td>Distance from CoG to rear axle</td>
<td>1.036 m</td>
</tr>
<tr>
<td>( M )</td>
<td>Vehicle mass</td>
<td>683 kg</td>
</tr>
<tr>
<td>( I )</td>
<td>Vehicle yaw inertia</td>
<td>561 kg \cdot m(^2)</td>
</tr>
<tr>
<td>( C_d )</td>
<td>Drag coefficient</td>
<td>0.5</td>
</tr>
<tr>
<td>( A_r )</td>
<td>Vehicle frontal area</td>
<td>4 m(^2)</td>
</tr>
<tr>
<td>( \rho )</td>
<td>Air density at 25°C</td>
<td>1.2 ( \frac{kg}{m^3} )</td>
</tr>
<tr>
<td>( \mu_0 )</td>
<td>Nominal friction coefficient</td>
<td>0.5</td>
</tr>
<tr>
<td>( C_x )</td>
<td>Tyre stiffness coefficient</td>
<td>15000 ( \frac{N}{rad} )</td>
</tr>
</tbody>
</table>

Table 2: Kinematic and dynamic model parameters

\[
\begin{align*}
\dot{x} &= v \cos(\theta) \\
\dot{y} &= v \sin(\theta) \\
\dot{\theta} &= \omega 
\end{align*}
\]  

where \( x, y \) and \( \theta \) represent the current position and orientation of the vehicle in meters (m) and radians (rad), respectively, with respect to the inertial frame \( \{W\} \). \( v \) is the linear velocity measured in \( \frac{m}{s} \) and \( \omega \) represents the vehicle angular velocity in \( \frac{rad}{s} \).

3.2.2 Dynamic model

The dynamical behaviour of a vehicle is generally complicated to represent in a detailed way. In practical applications, normally simplified models are used. In this case, the obtained model is based on the second Newton’s law. The streamlined dynamical model of the road vehicle can be written as
where $\alpha$ represents the vehicle slip angle (rad), $\delta$ is the steering angle and one of the inputs of the system (rad), $F_{xR}$ is the longitudinal rear force and the other input of the system (N), $F_{yR}$ represents the lateral rear force that appears when steering (N), $F_{yF}$ is the lateral front force which appears also with the angular motion (N), $F_{\text{drag}}$ represents drag force that opposes to the forward movement (N) and $F_{\text{friction}}$ is the friction force that also opposes to the longitudinal vehicle movement (N). In addition, note that instead of employing the states $x$ and $y$, a new representation has been adopted by using the polar representation and considers the variables $v$ and $\alpha$. These variables can be seen in Figure 8. Observe that the dynamic model variables are referred to the vehicle body frame $\{B\}$ while the kinematic set of variables refers to the global fixed coordinate system $\{W\}$ in order to represent the trajectory from a relative point of view.

### 3.2.2.1 Calculation of lateral forces using Pacejka’89 tyre model

Tyres are perhaps the most important, but difficult to model, component of an automobile. In addition to supporting the vehicle and damping out road irregularities, the tyres provide the longitudinal and lateral forces necessary to change the speed and direction of the vehicle. These forces are produced by the deformation of the tyre where it contacts the road during acceleration, braking and cornering.

In the absence of side forces, a rolling tyre travels straight ahead along the wheel plane. During a cornering maneuver, however, the tyre contact patch slips laterally while rolling such that its motion is no longer in the direction of the wheel plane (see Figure 9). The angle between its direction of motion and the wheel plane is referred to as the slip angle, $\alpha$. This lateral slip generates a lateral force, $F_{y}$, at the tyre-ground interface. Because the force acts slightly behind the center of the wheel, it produces an aligning moment, $M_{z}$, which tends to realign the wheel in the direction of rolling.

Normal cornering maneuvers result in small slip angles, low lateral force and minimal sliding of the tyre. At larger slip angles, lateral force increases and reaches a maximum as the tyre begins to slide. Figure 10 illustrates the relationship between lateral force and slip angle for a typical tyre. For small values of $\alpha$ (less than 4°), the relationship is nearly linear. The initial slope of the curve is known as the cornering stiffness, $C_{x}$, described in units of force per degree.

There exist many models to describe tyre behaviour beyond the linear region. One model commonly used in vehicle dynamics simulations was developed by H. Pacejka of the Delft University of Technology. The Pacejka tyre model calculates lateral force based on slip angle as follows:
The remaining model parameters of the Magic Formula are dependent on the normal force $F_z$ on the tyre where the normal force is given in kN and their definition can be found in Table 3.

For a further and more detailed description of the Magic Formula tyre model, the reader is referred to [14], where a complete listing of the model equations is given.

Finally, the front and rear tyre slip angles, $\alpha_F$ and $\alpha_R$, can be derived from the angular velocity $\omega$ and the translational components of velocity $v_x$ and $v_y$. 

\[ F_y = D \sin \left[ C \arctan \left\{ (1 - E)Bx + E \arctan(Bx) \right\} \right] + S_v \]  

(19)

with

\[ x = \alpha + S_h \]  

(20)
Table 3: Pacejka Magic Formula parameters

<table>
<thead>
<tr>
<th>$B$</th>
<th>stiffness factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>shape factor</td>
</tr>
<tr>
<td>$D$</td>
<td>peak value</td>
</tr>
<tr>
<td>$E$</td>
<td>curvature factor</td>
</tr>
<tr>
<td>$S_h$</td>
<td>horizontal shift</td>
</tr>
<tr>
<td>$S_v$</td>
<td>vertical shift</td>
</tr>
</tbody>
</table>

\[
\alpha_F = \delta - \arctan\left(\frac{v_y + a\omega}{v_x}\right) \tag{21}
\]
\[
\alpha_R = -\arctan\left(\frac{v_y - b\omega}{v_x}\right) \tag{22}
\]

with $v_x = v \cos(\alpha)$ and $v_y = v \sin(\alpha)$.

### 3.3 Dynamic LPV modelling

The LPV modelling task is presented in this section. This method consists on embedding the non-linearities inside model parameters that depend on some variables, called scheduling variables, that vary in a known bounded interval. In Section 3.2.2, dynamic non-linear model was presented. Here, an LPV representation of such model is introduced.

Denoting the state and control vectors, respectively, as

\[
x = \begin{pmatrix} v \\ \alpha \\ w \end{pmatrix}, \quad u_D = \begin{pmatrix} F_x R \\ \delta \end{pmatrix} \tag{23}
\]

the state space model for the dynamic representation (17) can be obtained as

\[
\dot{x} = A(\delta, v, \alpha)x + B(\delta, v, \alpha)u_D \tag{24}
\]

where:

\[
A(\delta, v, \alpha) = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & a_{32} & a_{33} \end{pmatrix} \tag{25}
\]
\[ a_{11} = -\frac{F_{df}}{Mv} \]
\[ a_{12} = \frac{C_x (\sin(\delta) \cos(\alpha) - \sin(\alpha) \cos(\delta) - \sin(\alpha))}{M} \]
\[ a_{13} = \frac{C_x (a (\sin(\delta) \cos(\alpha) - \sin(\alpha) \cos(\delta)) - b \sin(\alpha))}{Mv} \]
\[ a_{22} = \frac{-C_x (\cos(\alpha) \cos(\delta) + \sin(\alpha) \sin(\delta) + \cos(\alpha))}{Mv} \]
\[ a_{23} = \frac{-C_x a (\cos(\delta) \cos(\alpha) + \sin(\alpha) \sin(\delta)) + C_x b \cos(\alpha)}{Mv^2} - 1 \]
\[ a_{32} = \frac{C_x (b - a \cos(\delta))}{I} \]
\[ a_{33} = \frac{-C_x (b^2 + a^2 \cos(\delta))}{Iv} \]

\[
B(\delta, v, \alpha) = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ 0 & b_{32} \end{pmatrix} \quad (26)
\]

\[ b_{11} = \frac{\cos(\alpha)}{M} \]
\[ b_{12} = \frac{C_x (\sin(\delta) \cos(\alpha) - \sin(\alpha) \cos(\delta))}{M} \]
\[ b_{21} = \frac{-\sin(\alpha)}{Mv} \]
\[ b_{22} = \frac{C_x (\cos(\alpha) \cos(\delta) + \sin(\alpha) \sin(\delta))}{Mv} \]
\[ b_{32} = \frac{C_x a \cos(\delta)}{I} \]

Note that, at this point, \( A \) and \( B \) are time varying matrices.

### 3.4 Odometry-based localization using LPV Kalman filter

Due to the lack of available sensors for measuring all states, i.e.: there is no one that measures the slip angle, the design of a state estimator has been considered in this section. The LPV Kalman filter tackles the problem of estimating the dynamic states of the vehicle.

The measurement model for the dynamical one presented in Eq. (24) considering the available sensors lead to consider the following output matrix
due to the lack of measuring of dynamic states as the slip angle and the estimated vector state is denoted as

$$\hat{x}_{DO} = \begin{pmatrix} \hat{v} \\ \hat{\alpha} \\ \hat{w} \end{pmatrix}$$ \hspace{1cm} (28)

Then, the state estimation will depend on the observer gain $L$ (matrix of size $3 \times 2$) and presents the form

$$\dot{\hat{x}}_{DO} = A(\delta, v, \alpha) \hat{x}_{DO} + B(\delta, v, \alpha) u_D + L (y - C \hat{x}_{DO})$$ \hspace{1cm} (29)

A model of the actual vehicle pose $x_k = (x' y' \theta')^T$ after executing the motion command $u_k = (\hat{v} \hat{w})^T$ at $x_{k-1} = (x y \theta)^T$ is

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{\hat{v}}{T_s} \sin(\theta) + \frac{\hat{w}}{T_s} \sin(\theta + \hat{\omega} T_s) \\ \frac{\hat{v}}{T_s} \cos(\theta) + \frac{\hat{w}}{T_s} \cos(\theta + \hat{\omega} T_s) \\ \hat{\omega} T_s \end{pmatrix}$$ \hspace{1cm} (30)

where $T_s$ is the sampling time.

The derivation of this expression follows from simple trigonometry. Details about the derivation of Eq.(30) can be found in Chapter 5 of [2].

### 3.5 Optimal solution to Matrix Ricatti Equation for KF implementation

Considering the vehicle dynamical and measurement discrete-time models (discrete-time system matrices can be obtained from their continuous-time counterparts using zero-order hold), where additive Gaussian random noise has been included in order to describe the uncertainty introduced by the state transition and observation

$$\begin{align*}
x_{k+1} &= A_d(\delta, v, \alpha)x_k + B_d(\delta, v, \alpha)u_k + w_k \\
y_k &= Cx_k + v_k
\end{align*}$$ \hspace{1cm} (31)

the state estimation problem can be transformed into the following optimization problem:

$$\min_{\hat{x}_0, \ldots, \hat{x}_N} J = \sum_{k=0}^{N-1} w_k^T Q^{-1} w_k + \sum_{k=0}^{N} v_k^T R^{-1} v_k + w_0^T P^{-1} w_0$$ \hspace{1cm} (32)
\begin{align*}
\begin{cases}
    w_k = \hat{x}_{k+1} - (A_d \hat{x}_k + B_d u_k) & k = 0, \ldots, N - 1 \\
    v_k = y_k - C \hat{x}_k & k = 0, \ldots, N \\
    w_0 = x_0 - \hat{x}_0
\end{cases}
\end{align*}

where the design matrices $Q$ and $R$ are the expected covariance of disturbances and noise respectively, that is to say: $Q = E(ww^T)$ and $R = E(vv^T)$. Note that, in the optimal case, $w$ and $v$ are white noise processes with zero mean.

The previous optimization problem can be solved analytical way resulting in predictive Kalman filter, which consists in the following forward recursion:

\begin{align*}
\begin{cases}
    L_k = A_d P_k C^T (R + CP_k C^T)^{-1} \leftarrow \text{Observer gain} \\
    \hat{x}_{k+1} = A_d \hat{x}_k + B_d u_k + L_k (y_k - C \hat{x}_k) \leftarrow \text{Kalman Filter} \\
    P_{k+1} = Q + (A_d - L_k C) P_k A_d^T \leftarrow \text{Ricatti equation}
\end{cases}
\end{align*}

where $P_k$ is the covariance matrix of the optimal state estimation obtained from Ricatti equation at discrete-time step $k$.

If all noise is Gaussian, the Kalman filter minimizes the mean square error of the estimated parameters. In case the noise is not Gaussian, given only the mean and standard deviation of noise, the Kalman filter is the best linear estimator. Kalman Filtering has been widely used because it is suitable for online real time parameter estimation and measurement equations need not be inverted.
4 Solution based on odometry and LPV Kalman filter

The Linear Parameter Varying (LPV) observer tackles the problem of estimating the dynamic states of the vehicle. This section presents an LPV Kalman filter with both, stability and performance criteria, integrated in a single design procedure.

4.1 LPV Kalman filter design based on polytopic approximation

To design the observer gain $L$ in Eq.(29), a polytopic approximation of (24) is used

$$A(\Phi) = \sum_{i=1}^{2^{n_{\Phi}}} \mu_i(\Phi) A_i$$  \hspace{1cm} (35)$$

where $A_i$ are obtained using the bounding box approach, $n_{\Phi}$ is the number of scheduling variables, $\Phi$ is the vector containing the scheduling variables defined as $\Phi := [\theta_1(t), ..., \theta_{n_{\Phi}}(t)]$ and $\mu_i(\Phi)$ is given by

$$\mu_i(\Phi) = \prod_{j=1}^{n_{\Phi}} \xi_{ij}(\eta_j^0, \eta_j^1), \quad i = \{1, ..., 2^{n_{\Phi}}\}$$  \hspace{1cm} (36)$$

$$\begin{cases}
\eta_j^0 = \bar{\theta}_j - \theta_j(t) \\
\eta_j^1 = 1 - \eta_j^0,
\end{cases} \quad j = \{1, ..., 2^{n_{\Phi}}\}$$  \hspace{1cm} (37)$$

where each variable $\theta_j$ is known and varies in a defined interval $\theta_j(t) \epsilon [\underline{\theta}_j, \bar{\theta}_j]$.

Then, the observer gain is given by

$$L(\Phi) = \sum_{i=1}^{2^{n_{\Phi}}} \mu_i(\Phi) L_i$$  \hspace{1cm} (38)$$

where $L_i$ are the polytopic observer gains.

4.2 Calculation of polytopic observer gains using LMIs

The polytopic observer gains in Eq.(38) are obtained using the following proposition that provides an optimal design based on the Ricatti equations of the Kalman filter:

Proposition 1. "Let the observer tuning parameters $Q = Q^T \geq 0$, $R = R^T > 0$, the optimal performance bound $\gamma > 0$, the decay rate $\lambda > 0$, the output matrix $C$ in (27) and the matrices $A_i$ in (35). Then the polytopic observer gains $L_i$ are obtained by finding $Y$ and $W_i$ satisfying the following LMIs"

$$\begin{pmatrix}
Y A_i + A_i^T Y - W_i C - C^T W_i^T + Y 2\lambda & Y (Q^{1/2})^T W_i \\
Q^{1/2} Y & -I & 0 \\
W_i^T & 0 & -R^{-1}
\end{pmatrix} < 0$$  \hspace{1cm} (39)$$
\[
\begin{pmatrix}
\gamma I & I \\
I & Y
\end{pmatrix} > 0 \quad i = \{1, \ldots, 2^n\}
\]  (40)

considering \( Y = Y^T > 0 \) and applying the transformation \( L_i = Y^{-1}W_i \).

### 4.3 Dynamic LPV Kalman filter design

Then, the LPV Kalman filter addresses the problem of estimating the dynamic state vector in Eq. (28). At this point, the LPV model developed in (24) is used for solving the Proposition 1 using the output matrix (27). The chosen scheduling variables are \( \sigma, v \) and \( \alpha \) bounded in the following intervals

\[
\begin{align*}
\delta &\in [-25, 25] ^\circ \\
v &\in [0.07, 18] \frac{m}{s} \\
\alpha &\in [-0.1, 0.1] \text{ rad}
\end{align*}
\]

The proposed design matrices and parameters are: \( R = 0.01I_{2\times2}, Q = 0.01I_{3\times3}, \gamma = 50 \) and \( \lambda = 0 \). The solution of such a Proposition 1 returns the polytopic observer gains. Then, at every time step the interpolated observer gain is obtained by means of (38).

### 4.4 Pose integration

The actual vehicle pose \( x_k = (x', y', \theta')^T \) after executing the motion command \( u_k = (\hat{v} \hat{\omega})^T \) at \( x_{k-1} = (x \ y \ \theta)^T \) can be determined

\[
\begin{pmatrix}
x' \\
y' \\
\theta'
\end{pmatrix} = 
\begin{pmatrix}
x \\
y \\
\theta
\end{pmatrix} + \begin{pmatrix}
\hat{v} \cos(\theta + \hat{\alpha})T_s \\
\hat{v} \sin(\theta + \hat{\alpha})T_s \\
\hat{\omega}T_s
\end{pmatrix}
\]  (41)

Both velocities are kept at a fixed value for the entire time interval \((k - 1)T_s, kT_s\), where the sampling time \( T_s = 1 \text{ ms} \).

### 4.5 Comparison between LPV and Extended Kalman filter

Finally, we will stress the similarities and differences between LPV and Extended Kalman filters. Both methods assume Gaussian noise and deal with non-linear models. However, while classical Extended Kalman Filter approach linearizes around the estimated operating point, its LPV counterpart embeds the non-linearities inside model parameters that are allowed to vary in a known bounded interval resulting in a linear model.

On the other hand, EKF technique solves Ricatti equation online, whereas proposed LPV Kalman Filter approach solves Ricatti equation offline for vertex systems and performs online interpolation to determine filter gain for the current operating point.

Finally, LPV Kalman Filter shows additional benefits. It can cope better with systems with strong non-linearities due to the philosophy underlying the design procedure. Moreover, computational complexity is reduced because Jacobians are no longer used and it is enough to evaluate the LPV model matrices at the current operating point.
5 Integration of LPV Kalman Filter localization algorithms in SLAM Toolbox

The purpose of this section is to describe the basic features of the employed SLAM Toolbox for Matlab (see Section 5.1), where the developed state estimation techniques based on LPV Kalman Filter have been integrated for further use. Matlab implementation of the proposed vehicle models and the LPV Kalman Filter estimation approach is explained in Section 5.2. Finally, details about how the localization algorithm based on LPV Kalman Filter and pose integration has been included into the provided SLAM Toolbox are given in Section 5.3.

5.1 Introduction to SLAM Toolbox in Matlab

The Simultaneous Localization and Mapping (SLAM) problem asks if it is possible for a mobile robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map. The two traditional approaches of solving the SLAM problem is through the use of the extended Kalman filter (EKF-SLAM) and through the use of Rao-Blackwellised particle filters (Fast-SLAM). Graph-SLAM is another approach of solving the SLAM problem by introducing a graph.

The authors provide the interested user with two toolboxes sharing a number of features, but using two radically different approaches to EKF-SLAM, and nonlinear optimization via graphical models. They are provided for free under the GPL license and are available online at: https://www.iri.upc.edu/people/jsola/JoanSola/eng/toolbox.html.

**EKF-SLAM:** The first toolbox performs 6DOF SLAM using the classical EKF implementation. It is conceived as an “active-search” SLAM with points and with lines. It works with any number of robots and sensors. Monocular and stereo systems are treated alike.

**Graph-SLAM:** The second toolbox substitutes the EKF by a non-linear optimizer based on factor graphs and matrix factorization. Currently, QR, Cholesky, and Schur factorizations are implemented. It works with Euclidean points.

In a typical SLAM problem, one or more robots navigate an environment, discovering and mapping landmarks on the way by means of their onboard sensors. Robots of different kinds may exist, carrying a different number of sensors of different kinds, which gather raw data and, by processing it, are capable of observing landmarks of different kinds. All this variety of data is handled by this toolbox in a way that is quite transparent. 3D graphics utilities highly customizable are also provided with the SLAM toolbox.

For further details on the SLAM Toolbox for Matlab, the reader is referred to [16], which is focused on EKF-SLAM implementation. A complete tutorial on Graph-SLAM can be found in [17].

5.2 Matlab implementation of LPV Kalman Filter and dynamic vehicle model

The proposed approach to autonomous vehicle localization problem is based on the use of an LPV Kalman filter to estimate the vehicle dynamic state variables, which are used subsequently for pose integration purposes.
For the sake of completeness, the full SLAM Toolbox with the extension we have carried out will be provided together with this document. All Matlab files described in Table 4 are contained in a folder named TFM for easiness of localization.

<table>
<thead>
<tr>
<th>Matlab filename</th>
<th>Contents description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vehicleModelParameters.m</td>
<td>This function returns a structure with kinematic and dynamic vehicle model parameters in Table 2.</td>
</tr>
<tr>
<td>ode45VehicleModel.m</td>
<td>Kinematic and dynamic vehicle model: differential equations (16) and (17).</td>
</tr>
<tr>
<td>tyreModelPacejka.m</td>
<td>Pacejka’89 tyre model for lateral force calculation.</td>
</tr>
<tr>
<td>dyn3.m</td>
<td>Vehicle kinematics and dynamics evolution: differential equations solver.</td>
</tr>
<tr>
<td>lpvDesignParameters.m</td>
<td>This function returns a structure with parameters related to LPV observer design process (section 4.3).</td>
</tr>
<tr>
<td>stateObserverDesign.m</td>
<td>LPV Kalman filter design for vehicle dynamic state estimation based on polytopic approximation using LMIs (sections 4.1 and 4.2).</td>
</tr>
<tr>
<td>getDynamicLpvModel.m</td>
<td>This function returns dynamic LPV model for a given value of the scheduling variables vector (section 3.3).</td>
</tr>
<tr>
<td>getWeights_3.m</td>
<td>Calculation of polytopic LPV model weights in Eq.(36).</td>
</tr>
<tr>
<td>lpvKalmanFilter.m</td>
<td>LPV Kalman filter for dynamic states estimation in Eq.(29).</td>
</tr>
<tr>
<td>plot_results.m</td>
<td>Function for results plotting purposes.</td>
</tr>
</tbody>
</table>

Table 4: Matlab files implementing autonomous vehicle localization techniques

### 5.3 Integration of vehicle localization algorithm in Graph-SLAM Toolbox

The integration of the functionalities described in Section 5.2 in the SLAM Toolbox is explained in the following. Since at the moment of extending the toolbox with the developed algorithms, EKF-SLAM approach was not working properly, Graph-SLAM toolbox was selected instead.

Functions within the SLAM Toolbox are organized in three levels, from most abstract and generic to the basic manipulations. The highest level, called High Level, deals exclusively with the structured data related to the SLAM problem, simulator and graphics, and calls functions of an intermediate level called the Interface Level. The interface level functions split the data structures into more mathematically meaningful elements, check objects types to decide on the applicable methods, and call the basic functions that constitute the basic level, called the Low Level Library.

The high level scripts and functions are located in the directory `slambt-graph/HighLevel/`. There are two main scripts that constitute the highest level, one for the code and one for the data:

- **slambt_graph.m**: the main script. It initializes all data structures and figures, and performs the temporal loop by first simulating motions and measurements, second estimating the map and localization (the SLAM algorithm itself), and third visualizing all the data.

- **userDataGraph.m**: a script containing the data the user must enter to configure the sim-
autonomous vehicle localization using state estimation techniques

It is called by slamtb_graph.m at the very first lines of code.

High-level functions exist to help initializing all the structured data. They are called by slamtb_graph.m just after userDataGraph:

\[
\begin{align*}
\text{createGraphstructures}() \\
\text{createSimStructures}() \\
\text{createGraphicsStructures}()
\end{align*}
\]

The main purpose of these functions is to take the data from userDataGraph.m, which is just what the user needs to enter, and create with them the more complete structures that the program will use.

The interface level functions are located in the directory `slamtb-graph/InterfaceLevel/`. The interface level functions interface the high-level scripts and structured data with the low-level functions and the plain data. These functions serve three purposes:

1. Check the type of structured data and select the appropriate methods to manipulate them.
2. Split the structured data into smaller parts of plain data.
3. Call the low-level functions with the plain data and assign the outputs to the appropriate fields of structured data.

Interface-level functions perform the different simulation, SLAM, and redraw operations. They are called inside the main loop:

**Simulator:**

\[
\begin{align*}
\text{simMotion}() & \ % \text{Simulate motions} \\
\text{simObservation}() & \ % \text{Simulated observations}
\end{align*}
\]

**SLAM:**

\[
\begin{align*}
\text{motion}() & \ % \text{Robot motion} \\
\text{correctKnownLmk}() & \ % \text{EKF-update of known landmarks} \\
\text{initNewLmk}() & \ % \text{Landmark initialization}
\end{align*}
\]

**Visualization:**

\[
\begin{align*}
\text{drawMapFig}() & \ % \text{Redraw 3D Map figure} \\
\text{drawSenFig}() & \ % \text{Redraw sensors figures}
\end{align*}
\]

Then, taken into account the above considerations about the toolbox organization, the integration procedure has been done in the following manner:
1. Script `slamrc.m` that adds all subdirectories in `slamtb-graph/` to the Matlab path was properly extended.

2. A new user-data script named `userDataGraphReal.m` was created. Initially, it consisted in a copy of original user-data script `userDataGraph.m`. In a further step, it was modified to define a new robot structure for the urban vehicle and create an empty world of the appropriate dimensions.

3. A new main script named `slamtb_graph_real.m` was created. Initially, it consisted in a copy of original main script `slamtb_graph.m`. In a further step, it was modified to call the new user-data script and to include data logging and post-processing code. Those code lines related to graph solving have been commented, since at the moment no landmarks are used.

4. Interface-level function `createRobots()` has been extended to include a new motion model called `dynamic`. Basically, it re-organizes the information contained in the robot structure defined in the user-data script in a way more suitable for further processing within the SLAM algorithm.

5. Interface-level function `simMotion()` has been extended to include the implementation of a new motion model called `dynamic`. Each step of the dynamic loop consists of vehicle motion, LPV Kalman filter estimation of dynamic state variables and pose integration from the estimated slip angle and linear and angular velocities.
6 Simulation results

The simulation scenario chosen for testing the autonomous vehicle localization strategy is shown in Figure 11. Figure 12 show the control actions that will be used as inputs of the system. Note that this approach can be regarded as open-loop motion control (trajectory-following), because the measured robot position is not fed back for velocity or position control. Longitudinal rear force, $F_{x_R}$, and steering angle, $\delta$, input commands are kept at a fixed value for the entire time interval $((k-1)T_s, k]$, where the sampling time is equal to $T_s = 0.1$ s.

![Vehicle planar trajectory](image1)

Figure 11: Proposed circuit for simulation and the result of solving the localization problem

![Longitudinal rear force](image2)

Figure 12: Control actions used as system inputs: a) longitudinal rear force, b) steering angle.

The results of applying the LPV Kalman filter for estimating the vehicle dynamic states are shown in Figure 13 and 14. Ground truth has been considered to be the values of the dynamic vehicle model variables. The sample time used in dynamic loop is $T_s = 1$ ms.
Estimated vehicle linear and angular velocities, $v$ and $\omega$, and slip angle, $\alpha$, are employed subsequently for pose integration purposes. Figure 11 depicts the trajectory followed by the vehicle together with the estimated vehicle two-dimensional planar coordinates $x$ and $y$.

Finally, Figure 15 shows vehicle localization errors referred to the global fixed coordinate system $\{W\}$. In our results, the maximum error in the longitudinal direction is $x_{e\text{max}} = 1.33$ m. It can be observed that the vehicle experiments a pronounced acceleration stage during 15 seconds from the first 50 seconds. At this point, the vehicle orientation error in the z-axis was of about $0.3^\circ$ (see Figure 16). Thus, pose integration error accumulates and the localization algorithm no longer recovers from it. Error along y-axis remains in the scale of few decimeters (maximum error in the lateral direction is $y_{e\text{max}} = 0.55$ m).
Figure 13: Results of applying the dynamic LPV Kalman filter for solving the localization problem: a) linear velocity, b) angular velocity.

Figure 14: Results of applying the dynamic LPV Kalman filter for solving the localization problem: slip angle.
Figure 15: Position errors referred to global fixed coordinate system $\{W\}$.

Figure 16: Vehicle orientation error in the z-axis.
7 Conclusions and future work

This master thesis has explored the use of an LPV Kalman filter for estimating the position of an autonomous vehicle based on odometry using a kinematic/dynamic model.

This method represents an innovation with respect to the existing localization methods because most of the existing works in mobile robotics consider only a kinematic model since the velocity is very low. However, in autonomous vehicle applications where the velocity is higher, a dynamic model is required as already has been proved in the design of controllers for trajectory tracking.

The use of EKF is the predominant approach in autonomous vehicle localization. However, EKF is based on a linearization around the estimated trajectory while LPV Kalman filter does not use linearization but instead embeds the non-linearities in the varying parameters. Moreover, the design of the LPV Kalman filter is based on the use of LMIs and Lyapunov theory providing guarantees of convergence (stability) of the estimations. Finally, the LPV Kalman filter does not require solving the Ricatti equation on-line but instead being possible to be solved off-line in the region of varying parameters.

After revising the problem of localization using odometry using the classical approach based on EKF in Chapter 2, it is reformulated considering that vehicle model includes both the dynamic and kinematic parts in Chapter 3. The non-linear vehicle of the model is reformulated in a LPV way by means on the non-linear embedding approach and the polytopic representation obtained using the bounding box approach. Once the vehicle model is formulated in the polytopic LPV form, in Chapter 4, the LPV Kalman filter is designed by solving the Ricatti equations formulated in the LMI form in the vertices of the polytope embedding the region of parameter variation. A set of Kalman filters gains are obtained as well as the interpolation rules that will allow the on-line implementation by scheduling the observer according to the operating point. The proposed approach has been integrated in the SLAM toolbox in Chapter 5, a real autonomous vehicle is simulated using the non-linear simulator also embedded in the toolbox. Using this set-up, simulations results are obtained that validated the proposed localization approach.

However, the results still show space for improvement. In particular, a better integration between the kinematic and dynamic parts should be developed. This can be achieved by designing an observer that integrates both the dynamic and kinematic parts. In the current version of the observer estimates only the dynamic states while the kinematic ones as obtained by means of numerical integration. Another extension that is planned is the inclusion of landmarks in the LPV Kalman observer and the map building capability.
Acknowledgments

I would like to dedicate my master thesis to my grandfather Pau, who died without fulfilling his dream of graduating from college, and to my uncle Alfons for his unconditional love and support wherever he is. Finally, I would like to thank my supervisor, Prof. Vicenç Puig, for his patience, help, and the numerous and fruitful discussions throughout carrying out this work.
Bibliography


